# Virtual executive toys
# Hybrid systems on the internet

A dissertation submitted to the University of Manchester

for the degree of Master of Science

in the Faculty of Engineering and Physical Sciences

**2011**

**NIKOLAS DIMITRIOU**

**School of Computer Science**

# Table of contents


**1. Introduction ..... 7**

1.1. Aims ..... 7

1.2. Objectives ..... 8

**2. Literature – Dynamical Systems ..... 9**

2.1. Continuous systems ..... 10

2.2. Discrete systems ..... 11

2.3. Hybrid Systems ..... 12

2.3.1. Bouncing ball ..... 13

2.3.2. Thermostat – control system ..... 13

2.3.3. Balancing robots - Inverted Pendulum ..... 14

**3. Technical background – Dynamic web pages ..... 16**

3.1. Client – server architecture ..... 16

3.1.1. Web 1.0 ..... 17

3.1.2. Web 2.0 – Ajax introduction ..... 17

3.2. Dynamic Updating ..... 17

3.2.1. Ajax ..... 18

3.2.2. XMLHttpRequest ..... 19

3.2.3. CSS ..... 20

3.2.4. DOM ..... 22

3.3. Web services ..... 23

3.3.1. SOAP Vs REST ..... 23

3.3.2. SOAP ..... 25

3.3.3. WSDL ..... 26

3.4. Drawing simulations ..... 28

3.4.1. Canvas ..... 28

3.4.1.1. Explorer canvas ..... 28

3.4.1.2. Canvas initialization ..... 29

3.4.1.3. Canvas shapes ..... 30

3.4.2. WebGL ..... 32

3.4.2.1. Initialize WebGL ..... 33
3.4.2.2. Draw with WebGL ..... 35
**4. System functions ..... 37**
4.1. Use case diagram ..... 38
4.2. Functions' description ..... 39
4.2.1. "Select application" Use Case ..... 39
4.2.2. "Dimension Reduction – add/remove point" Use Case ..... 40
4.2.3. "Dimension Reduction – view point coordinates" Use Case ..... 41
4.2.4. "Dimension Reduction – zoom in/out graph axis" Use Case ..... 42
4.2.5. "Single pendulum – start/pause simulation" Use Case ..... 42
4.2.6. "Single pendulum – change initial condition" Use Case ..... 44
4.2.7. "Double pendulum – start/pause/restart simulation" Use Case ..... 45
4.2.8. "Double pendulum – change initial condition" Use Case ..... 46
4.2.9. "Double pendulum – Manually drive simulation" Use Case ..... 48
4.2.10. "Double pendulum 3D – enable/disable autorotation" Use Case ..... 49
4.2.11. "Double pendulum 3D – Change the viewing angle" Use Case ..... 50
4.2.12. "Edit graph properties – Change graph attributes" Use Case ..... 51
4.2.13. "Edit graph properties – Change graph appearance" Use Case ..... 52
4.2.14. "Edit graph properties – enable/disable autozoom property" Use Case 53
4.2.15. "Edit graph properties – manually zoom in/out graph axes" Use Case 54
4.2.16. "Edit graph properties – move graph coordinates on X axis" Use Case 55
4.2.17. "Edit graph properties – clear graph" Use Case ..... 56
**5. System and Technology Architecture ..... 57**
5.1. Model of the problem ..... 57
5.2. System Architecture ..... 58
5.3. Technology Architecture ..... 59
5.3.1. Client Side ..... 60
5.3.2. Server side ..... 60
**6. Implementation ..... 63**
6.1. Core Functionality ..... 63
6.1.1. Client – side ..... 63

6.1.2. Server – side ..... 63
6.2. User Interface ..... 64
Main menu ..... 64
6.2.1. Dimension Reduction application ..... 65
6.2.2. Single pendulum application ..... 69
6.2.2.1. The options panel ..... 69
6.2.2.2. Draw simulation ..... 70
6.2.3. Double pendulum application (2D) ..... 73
6.2.3.1. New features ..... 73
6.2.3.2. Geometry of the double pendulum ..... 76
6.2.3.3. Drawing the simulation ..... 77
6.2.4. Double pendulum application (3D) ..... 80
6.2.5. Graph Plotter ..... 83
**7. System tests ..... 88**
7.1. Load application ..... 88
7.2. Dimension Reduction application ..... 88
7.3. Single pendulum application ..... 89
7.4. Single pendulum application ..... 89
7.5. Double pendulum application (2D) ..... 90
7.6. Double pendulum application (3D) ..... 91
7.7. Graph plotter ..... 92
**8. System evaluation ..... 95**
**9. Limitations – Future work ..... 97**
**10. Project plan ..... 98**
**11. Summary ..... 98**

Final world count: 20,990

## Abstract:

**Dynamical systems** are systems which their state is changing over time. They can be divided into three categories. **Continuous systems** experiencing a continuous behaviour, **Discrete systems** where their state can select between a group of values and **Hybrid Systems** which include both continuous and discrete components. Hybrid systems can be the result of a physical system when it is being controlled using a digital system. Such example is a thermostat which consists of a heater (discrete part – on/off) and the room temperature (continuous part). CICADA group has designed and implemented several hybrid system models related to their research domain. For example an inverted pendulum, digitally controlled to keep an upright position, can be used to describe the human failing aspect of the human body and thus the balance of human - like robots.

The purpose of this project is to represent the hybrid systems designed by CICADA members as a web interface in order to give the opportunity for new students to study and practise the activity of those systems. Simulated graphic representations demonstrate the behaviour of the implemented systems. Users are able to adjust the parameters related to the activity of the systems in order to see how they affect the system's run. Important task of the application is to display the new state of the system dynamically.

In order to accomplish the dynamic updating feature of the web application, the Ajax technologies have been implemented including JavaScript, XMLHttpRecuest, Document Object Model (DOM), and Cascading Style Sheets (CSS). The server-side component of the system has been implemented using Java Server Pages (JSP) and it is be responsible to execute dynamical systems' simulators indicating the behaviour of the different systems. The communication between server and client sides has been implemented using the XMLHttpRequest object of Ajax which is able to forward synchronous and asynchronous request on the server.

## Declaration Statement

No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

## Copyright Statement

I. Copyright in text of this dissertation rests with the author. Copies (by any process) either in full, or of extracts, may be made **only** in accordance with instructions given by the author. Details may be obtained from the appropriate Graduate Office. This page must form part of any such copies made. Further copies (by any process) of copies made in accordance with such instructions may not be made without the permission (in writing) of the author.

II. The ownership of any intellectual property rights which may be described in this dissertation is vested in the University of Manchester, subject to any prior agreement to the contrary, and may not be made available for use by third parties without the written permission of the University, which will prescribe the terms and conditions of any such agreement.

III. Further information on the conditions under which disclosures and exploitation may take place is available from the Head of the School of Computer Science.

# 1. Introduction

Hybrid systems of different models are being developed by the Research group of the CICADA project. A hybrid system appears as an outcome of a physical system when it is being controlled through a digital system. The definition of hybrid systems describes them as dynamic systems that reveal both discrete and continuous dynamic behaviour. The term “dynamic system” is used to describe systems with a changing state over time. An example of a hybrid system is the thermostat controlling a room’s temperature; it comprises of the room temperature (continuous behaviour) and the heater (discrete behaviour).

Beyond the aforementioned simple thermostat example, CICADA project has developed a range of different hybrid systems. A central concern is to analyse the behaviour that such systems implement, and apply new methods that intend to verify and improve the control of different hybrid systems.

## 1.1. Aims

One area of CICADA is responsible to provide training for new students and researches. The scope of this project is to improve the way of teaching and training of new members interested to be involved in the area of hybrid systems. Students need to be familiar with the way that the implemented systems respond while altering their key parameters which are affecting the operation of those systems. Experiencing the system’s responses students will obtain a better understanding of the system’s operation and how to control the system using those parameters.

However practical simulation and testing on those systems requires a lot of user training and preparation time beforehand. Also because of the unexpected behaviours of several systems, sometimes it might be difficult for a new student to attempt it by its own. For this particular reason this project attempts to extend the way students experiment on hybrid systems with the introduction of a web-based interface which represents simulations of different hybrid models. The interface will be accessible online for students and thus by adjusting the system’s parameters and observing its responses they would be able to practise the dynamical systems on the internet.

## 1.2. Objectives

CICADA implemented some programs representing the behaviour of the hybrid systems. Almost all of the implementations are running using C++ or MATLAB environments. Those programs are taking some inputs representing the parameter changes of the system and returning the new state of the system as output. Main task of the project was to create a web-page implementation which interacts with the already existing programs, passing the input parameters and obtaining the given output after the execution of the program. Input parameters are being entered by the users of the web-page (i.e. students) and the output of the program is being displayed to the user through the web-side instantly.

The web page representation of the hybrid system needs to be an easy to use as well as a user-friendly implementation. Fields for parameter inputs and adjustments are available for controlling the system's actions. After the execution of the program with the given parameter inputs, outputs representing the system's responses are visible through the webpage in the form of simple numeric values or in some cases a visualisation of the hybrid system is available.

Important project task is to use web tools for dynamically updating the web-page or particular fields on it without the need of reloading the page for every parameter change. This can be achieved using JavaScript, AJAX and DOM web developing tools, which provide the ability to use asynchronous requests running on the background and update different parts of the web-page without reloading. In this way the new state of the hybrid system representation is being updated dynamically on every parameter input.

# 2. Literature – Dynamical Systems

This chapter intends to provide some basic knowledge on dynamical systems. Since the application of this project has been designed to describe the behaviour of such systems, it is being clarified what dynamical and hybrid systems are and which are their key characteristics.

Every system can be described as dynamical when its state is changing over time. Dynamical systems are being determined using two attributes. The first one is the "state vector", which shows the state of the system at a given time, and the second one is the "system function", which is used to calculate the future state of the system based on the previous one. This function is being expressed as a mathematical statement **[1]**. Depending on the type of the system there is a variety of equations, from simple (i.e. linear) to very complex ones which can be used to describe the behaviour of different systems. Thus, by knowing an instance of the system's state vector and the system's function, someone can predict all the future states and behaviour of the system after that state.

The state of a dynamical system can always be described using numbers. This is due to the ability of all the physical systems to be descried using mathematical numbers **[1]**. A ball thrown upwards, for example, can be described with only two numbers, the velocity of the ball and its distance from the ground. However we cannot describe the state of every dynamical system by using just a few numbers. A dynamical system representing the climate conditions for instance, would need a huge amount of measurement attributes to describe its state. Although despite the complexity of a dynamical system, its state can still be described just by using a list of numbers (vector) no matter how big this list is **[1]**.

Dynamical systems can by separated into 3 different categories. Based on their state type and the time of the state change event, dynamical systems can be described as **continuous**, **discrete** or **hybrid** systems.

## 2.1. Continuous systems

A dynamical system is being described as continuous "if the state takes values in Euclidean space $R^n$ for some $n \geq 1$" **[2]**. Its state changes continuously through the time and not only at specific time intervals. Continuous systems are also divided into two subcategories:

1. **Linear:** the function describing the state changes is expressed as a linear differential or difference equation.
2. **Nonlinear:** the function describing the state changes is expressed as a nonlinear differential or difference equation.

**Pendulum**

A very frequent example of a continuous system is the pendulum. The pendulum consists of an object, usually a small ball, with some weight hanging from a fixed surface. If we force it to shift from its initial position, it will start swinging due to the gravity applied on it and finally return to its initial position again. There are different kinds of pendulums such as the regular pendulum, the double pendulum and the inverted pendulum. The pendulum is a nonlinear continuous system. A regular pendulum is shown in figure 2.1 below.

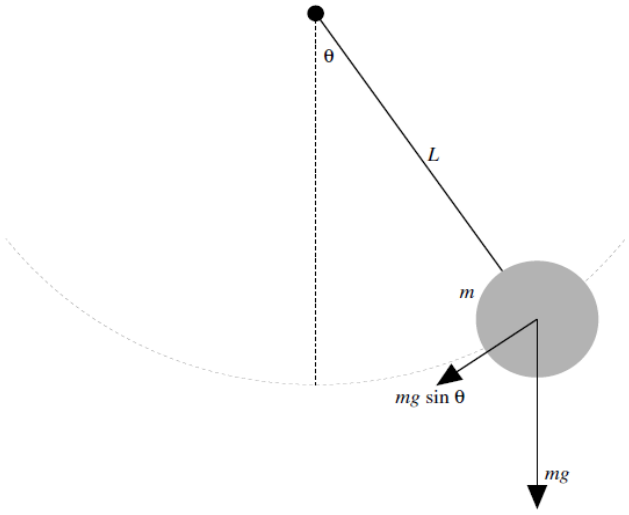



Figure 2.1: Regular pendulum

Since the application includes some pendulum implementations, this section consists of a small background on the attributes of the regular pendulum which are affecting its motion. The application contains different kind of pendulums either like animating

virtualisations of their movements or as a source for obtaining data and use them in other implementations. In both cases the user is able to control the run of the pendulums by adjusting the parameters which affect its movement.

A pendulum can be described by the following attributes:

**m:** maze of the object

**L:** length of the pendulum

**θ:** angle of the pendulum

**g:** gravity

**mg:** force of the gravity

**mg sin θ:** force applied on the pendulum causing its movement

Altering any of the attributes m, L, θ or g, if possible, will affect the run of the pendulum. The state of a pendulum can be described using the values θ (the angle of the pendulum) and ω ("the rate of rotation" [1]) which can be found by ω=dθ/dt.

Applying the Newton's law F= ma, we can result to the following equations:

1. θ'(t)= ω(t)
2. ω'(t) = -(g/L) sin θ(t)

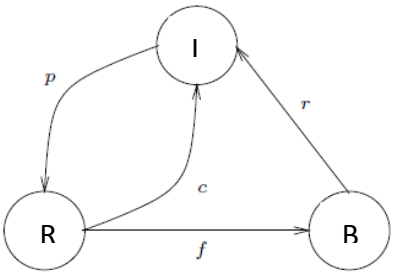
The next state x' of the pendulum can be found using the equation x'=f(x) where x is an already known state and f is the system function. Since the state vector x=$\begin{bmatrix}\theta\\ \omega\end{bmatrix}$, from the equations 1 and 2 above, we can result that the system function will be:

f $\begin{bmatrix}\theta\\ \omega\end{bmatrix}$= $\begin{bmatrix}\omega\\ -(g/L)\sin\theta\end{bmatrix}$ [1].

## 2.2. Discrete systems

Dynamical systems are described as discrete systems if their state can take values from a specific set only. Thus the states of the system are predefined and the system can jump from one state to another within the set. However in several cases the system cannot move from its current state to any of the remaining states. Such systems are controlled by different rules identifying which values the system can take based on its current state. To represent discrete systems we can use directed graphs or automata.

An example of a discrete system can be a simple switch which can take only two values, q ∈ {ON, OFF} or a machine which can jump between three available values, q ∈ {run, idle, broken} **[2]**. For the second example, we can set some rules to control the movement from one state to another. For example, the machine can only move to the run state after idle or it may be broken only while it was running. It can move from run back to idle or from broken mode to idle after being repaired. The representation of this machine using a directed graph will look like in the figure 2.2 below.




States: $q \in Q = \{I, P, B\}$
Symbols: $\sigma \in \Sigma = \{p, c, f, r\}$
Transitions: $\delta: Q \times \Sigma \rightarrow Q$

$\delta (I, p) = R$
$\delta (R, c) = I$
$\delta (R, f) = B$
$\delta (B, r) = I$

Figure 2.2: Discrete system - machine

Dynamical systems are described as discrete based on their type of state. There be also a categorisation of different systems based on the time interval between each state change. Dynamical systems are defined as **"discrete-time"** if their state evolves only after a fixed period of time. In contrast with discrete systems, discrete-time systems can take any value after every state change instead of a fixed amount of values. Example of a discrete-time system is the microchips of a computer which are changing their state after each clock cycle [1]. A dynamical system can be both a discrete and a discrete-time system.

## 2.3. Hybrid Systems

Hybrid systems are systems which express both a continuous and a discrete activity. One part of the systems changes its state value continuously while the other part will have a periodic change. A simple example of a hybrid system is the bouncing ball. The bouncing ball performs a continuous activity while it is moving up and down while its discrete part occurs each time the ball bounces on the ground **[3]**. In the figure 2.3 below we can see the directed graph on the left representing how the state of the ball changes its value each time it bounces on the ground.

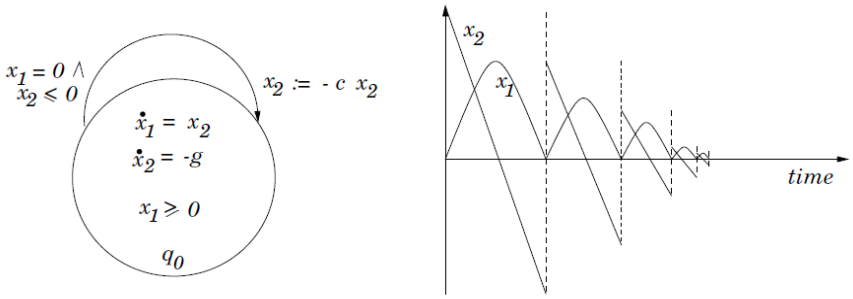



Figure 2.3 Bouncing ball

### 2.3.1. Bouncing ball

The graph on the right shows the process of a sample bouncing ball after its fall from some height. It describes the velocity (x1) and vertical position (x2) of the ball against time. As it has been mentioned earlier, the state of the system can be described using the values x1 and x2, thus x=$\begin{bmatrix} x1 \\ x2 \end{bmatrix}$.

From the directed graph on the left we can see the effect of the discrete part of the system. Discrete part (bounce) takes place when the velocity of the ball is zero and the vertical position is equal or less than zero (i.e. $x1 = 0 \wedge x2 \leq 0$). The position of the ball is represented with positive numbers on his way down and negative numbers on his way up. This is expressed by the equation x2:=-c x2 with $c \in [0,1]$ which gives the negative position of the ball after the bounce. After the bounce, the ball looses energy and therefore its velocity, until finally its velocity becomes zero.

### 2.3.2. Thermostat – control system

Hybrid systems usually occur during the usage of technology to control the behaviour of different physical systems. Such systems are often called **"control systems"**. A frequent case of control systems is the thermostat. Thermostat consists of both continuous and discrete state components. As continuous state pat we consider the temperature of the room while the discrete state component consists of the heater which can take any of the two values on or off ($q \in$ [on, off]) **[4]**. In figure 2.4 we can see the state automaton representing a thermostat system keeping the temperature of the room at approximately 75°C.

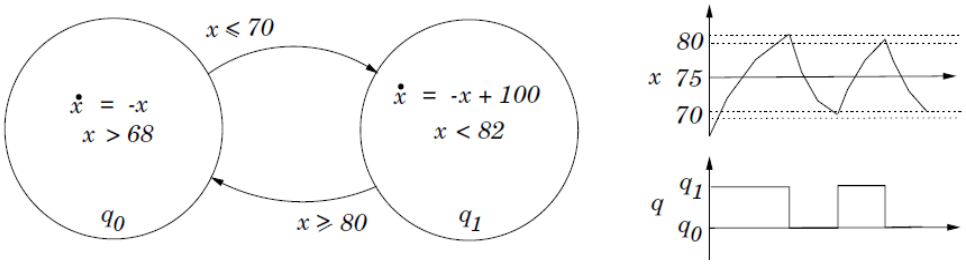



Figure 2.4: Thermostat

The thermostat is turning the heater on when the temperature is below 70°C and off if the temperature exceeds 80°C. The values x>68 and x<82 at states q0 and q1 respectively correspond to the temperature values on each stage as temperature may fall up to 68°C after the heater has been turned on and likewise the temperature may go up to 82°C after the heater is turned off. The upper graph in figure 2.4 represents the temperature value (continuous component) against time during a sample run of the thermostat while the lower graph shows the behaviour of the heater (discrete component).

### 2.3.3. Balancing robots - Inverted Pendulum

One of the CICADA researching sections is mainly concerned in solving control problems. Control problems occur when we are trying to control the behaviour of physical systems. The "**control systems"** as described above represent the solution of such problems. As a control problem example we can consider the balancing problem of bipedal walking robots.

Balancing on human beings is quite similar to the control problem of a balancing robot. Human system controls the human body to be in a straight position applying small forces pushing the body forward or backward keeping it still **[5]**.

We can associate the balancing problem on robots with the control of an inverted pendulum **[6]**. The inverted pendulum is adjusted on a sliding object which can shift in only two directions (i.e. forward or backward) as shown in the figure 2.5 below. Adjusting its movement regarding to the falling side of the pendulum, we can control and thus stabilize the pendulum in a continuous straight position.

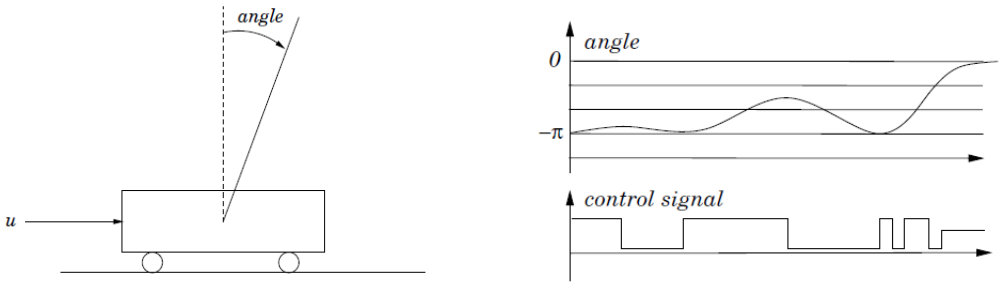



Figure 1.5: Inverted pendulum

The inverted pendulum is a hybrid system with the falling pendulum representing its continuous component and the sliding object expressing a discrete behaviour. A sample run of the inverted pendulum is described by the two graphs in figure 2.5. The angle of the pendulum shows its position at each point and thus its continuous behaviour while the control signal defines its discrete activity.

The last example illustrates the importance of the dynamical systems and why CICADA developers are studying those systems trying to discover new methods to control their behaviour.

# 3. Technical background – Dynamic web pages

This chapter contains information on what programming methods and tools are required in order to implement an application covering the objectives as set in chapter1. Main concern of the implementation is to provide the ability of dynamic updating web page information creating an efficient as well as easy to use application. Important aspect in developing dynamic updated web pages is the use of Ajax technology. Ajax has been introduced for this very reason; i.e. to provide an efficient way for representing information in a web application covering all the required functionality for dynamic updating.

## 3.1. Client – server architecture

Client – server architecture is a distributed model where each network part is described either as client or server. Servers are computers running on the network as hosts providing their services and resources upon client requests. There are different types of servers based on the services they provide, i.e. database servers, file servers or printer servers. Each server may be connected to one or more servers to support its applications. Clients request services from different servers through their applications and proceed based on the server's response **[7]**. Web is built on the client-server architecture. Server computers are responsible for processing client's requests while clients receive the server's response and display the appropriate information through the browser. The client- server architecture model is shown in figure 3.1 below.

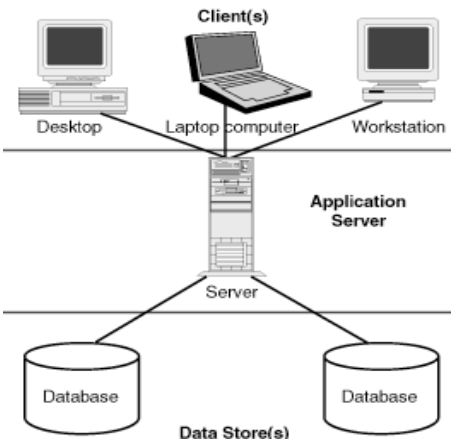



Figure 2.1: Client - server architecture

### 3.1.1. Web 1.0

The term Web 1.0 describes the first introduction of published web pages in the World Wide Web (WWW). Since the release of WWW for public use in 1993, people use the web to present their information through the internet **[8]**. Static web content using tables to align the contained information, and contact forms delivered via email were some of the usual features of the web pages created under Web 1.0. Users were just reading the information available through the web page since they were not able to interact with its content.

### 3.1.2. Web 2.0 – Ajax introduction

Web 2.0 or the “read/write” web as it has been described, created the ability for users of the web to interact with the content available inside the web pages. Users now do not only read the information displayed through the web sides but they can also participate and dynamically update their content **[9]**. “The concept of "Web 2.0" began with a conference brainstorming session between O'Reilly and Media Live International in 2003.” **[10]** Ajax became available through web 2.0 as many web developing tools made possible the use of asynchronous requests and dynamic modification of web content.

## 3.2. Dynamic Updating

Dynamically updating web pages presents a much more efficient way of interaction between the users and the information provided inside a web application. New technologies and interfaces introduced tools for dynamic updating. Web sites no longer need to use different URL address to represent each different stage of a page since individual parts of the web page can be updated without requiring the whole web page information to be reloaded (see figure 3.3). Many modern web applications are implemented using tools for dynamic updating their data. Applications like Google maps, flash implementations or chat interfaces embedded in websites would be more difficult or even impossible to be implemented without the use of dynamic updating functionality.

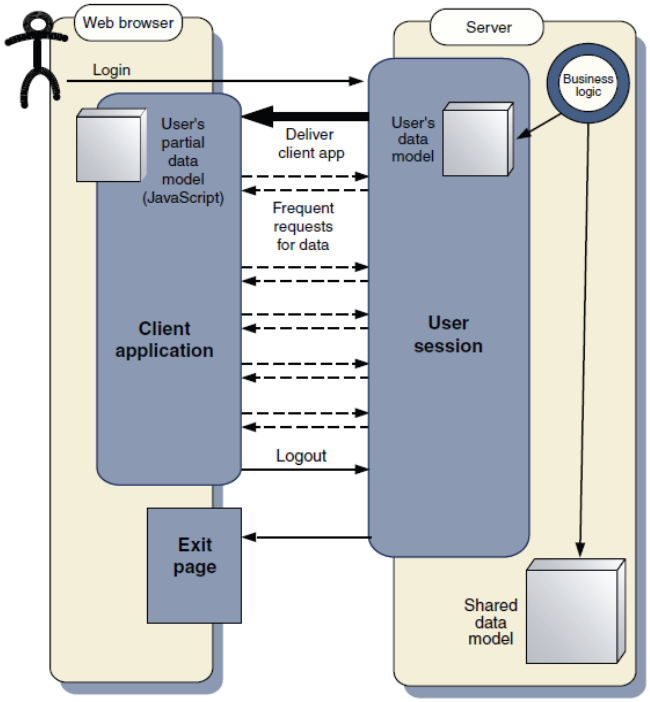



Figure 3.3: Web page using dynamic updating

### 3.2.1. Ajax

Ajax (Asynchronous JavaScript and XML) is not a new technology. It represents a combination of technologies made available through the introduction of Web 2.0 **[11]**. The first application of Ajax was designed by Google in 2005 and presented as what is known today as Google Suggest **[12]**. Google engine in nowadays is even more sophisticated with Google designers to prove how far we can get with Ajax. Search results are being displayed dynamically on user input while content is being automatically updated after each modification of the search key words.

As mentioned above, using the term Ajax we do not refer to a single technology. Ajax is a client-side implementation based on a collection of web design functionalities running together as shown in figure 3.4. Main concerns are the client-server communication running on the background and the efficient representation of new web-page information.

**Ajax** is implemented using the following components:

- **HTML/XHTML** with **CSS:** to design and style the web page information
- **XMLHttpRequest:** to create asynchronous requests
- **DOM (Document Object Model):** to update specific content of the browser
- **XML:** to send/receive data
- **JavaScript:** business logic - access and control the above technologies

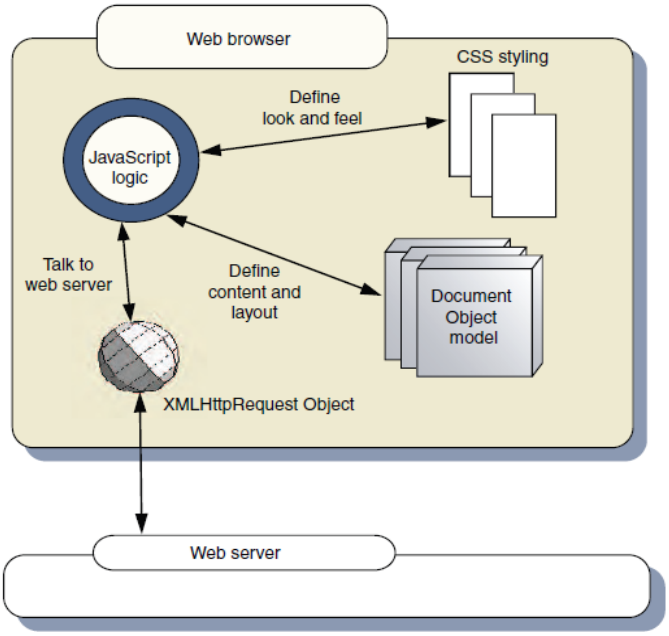



Figure 3.4 Ajax components

### 3.2.2. XMLHttpRequest

"XMLHttpRequest is a JavaScript object that was designed by Microsoft, adopted by Mozilla, and is now being standardized in the W3C" **[13]**. IT is considered as one of the most important parts of AJAX lifecycle as it provides the ability to send asynchronous requests to the server and receive its responses in the background. Regardless of the name, XMLHttpRequest is able to send and receive several data types beyond the XML. The more frequently used transport protocol of XMLHttpRequest is HTTP. However, based on its W3C specification, additional protocols like FTP and FILE are also supported **[14]**.

The XMLHttpRequest object is currently supported by the latest versions of all the modern internet browsers. It can be accessed through JavaScript language on the client-side. Using the URL of the server-side application, we can send a request to the server using any of the HTTP request methods (i.e. GET, POST, PUT or DELETE) **[15]**. Upon receive of the request, server-side application processes the request and generates the appropriate respond. The respond may be received in several formats such as a plain string or an XML Document. In the case of error during the communication, client will receive an error code identifying the type of the problem.

A sample call of the XMLHttpRequest object is shown below:

```
1. var req = new XMLHttpRequest();
2. req.open("GET","http://www.example.php",true,null,null);
3. req.send(null);
4. var response =req.response;
```

**Note:** in older versions of internet browsers the declaration of XMLHttpRequest object may vary. JavaScript can handle this possibility identifying the version of the browser in use and whether it supports the use of XMLHttpRequest.

The “req.open” command can take up to 5 parameters. The first parameter identifies the HTTP method of the command. The second parameter gives the URL of the server application to be called. The third parameter can take two values (i.e. true/false) to describe if we need an “asynchronous” (true) or “synchronous” (false) request. More precisely using asynchronous request, the application will not wait for the server’s response to carry on with its remaining processes while a synchronous request must receive the response first before calling the remaining functions. The default value of the third parameter is true. The last two parameters are optional to be used if the server-side application requires user authentication using username and password.

### 3.2.3. CSS

CSS (Cascading Style Sheets) were designed to provide the ability of styling in an HTML document as well as the use of the same styling format in multiple pages and elements. In addition, from maintenance view, the modification of style information in a web application became much more effective using CSS. The purpose of Cascading Style Sheets during the design of a web application can be divided into two tasks. The first one is to define the position and dimensions of the different elements as it offers different attributes which can be used to describe the elements’ layout. The second task is to add style (i.e. colour, transparency, images) in the displayed elements of the application **[11]**. A sample use of CSS is shown in the figure 3.5 below. The left picture displays the look of the application when we define the positioning attributes in CSS while in the right one is the view of the same application after the addition of the styling attributes.

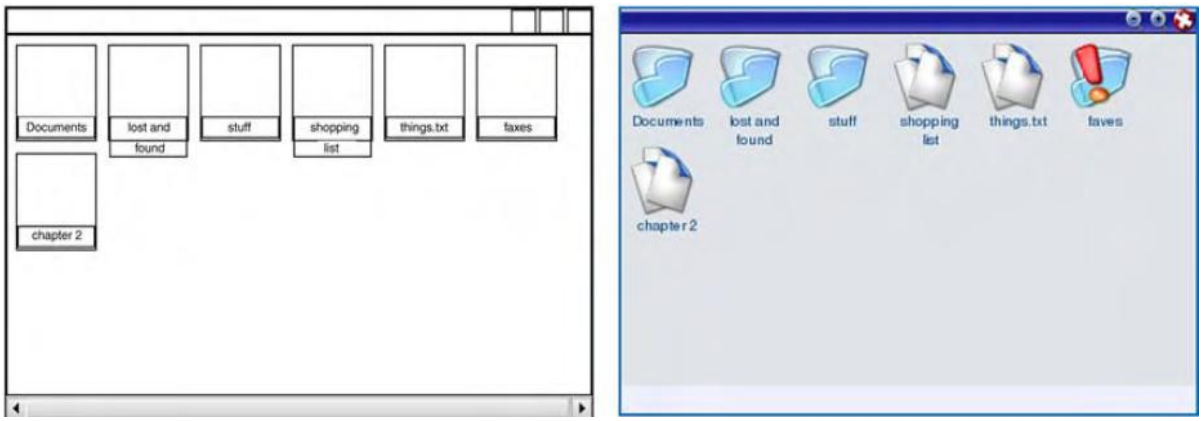



Figure 3.5 CSS: position and style

The use of CSS along with Ajax is even more powerful as Ajax provides the ability to select between different predefined element styles and apply them dynamically on the HTML elements. In this way we can design much more efficient as well as attractive applications.

Furthermore with CSS we can specify different styling rules which will be used to describe the style of our application when it is displayed on different machines such as laptops or mobile phones.

CSS are stored in a separate file containing all the styling information. In some cases it might be embedded in an HTML document but this will disable some of its basic features such as the reuse of same style sheet for more than one pages. CSS is described using simple rules in the form of “selector” - “style declaration” as shown in the example below.

```
h1 { color: blue }
```

The “selector” in this example is h1 and represents all the elements using the <h1> tag in document. The “style declaration” part is {color: blue} where we can include any style information we want to apply in different elements. For example, including this CSS rule in an HTML document we will adjust all h1 elements to appear in blue colour.

Apart from the element name, the selector can take several values describing specific parts of the HTML document. This may be a class name used in the HTML document or an ID name describing a single element. Possible “selector” values are shown in the following page **[16]**.

head {color: blue} ⟵ head is an element name <head>

.box { color: green} ⟵ box is a class name 

#first { color: white } ⟵ file is an element ID value 

a:hover{ color: red; } ⟵ dynamic value – displayed when the mouse is over element a

### 3.2.4. DOM

DOM (Document Object Model) is a tree style view of an HTML or XML document representing all the elements appeared in the document in a hierarchical order. A DOM tree example describing the CSS web application of the previous chapter is shown in figure 3.6. DOM actually provides the ability of dynamically updating and extending a web page since through it we can locate and manipulate any element included in the HTML document. Using JavaScript we can interact with the DOM document and update its content; we are able to read, modify or remove any property of the displayed elements or even create completely new elements and attach them inside the current document **[17]**.

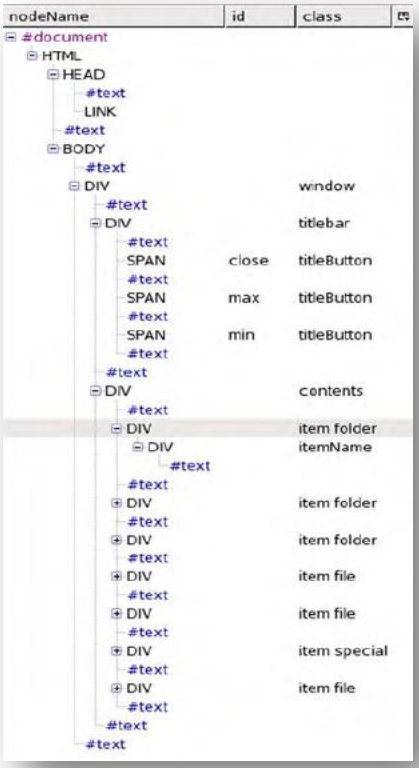



Figure 3.6: Document Object Model

A simple DOM access using JavaScript is shown below:

```
1.  var first = document.getElementById('first');  //get element using id
2.  first.color='blue';    //update style information
3.  first.background-color='white';
```

## 3.3. Web services

"A web service is any service that is available over the internet, uses a standardized XML messaging system, and is not tied to any one operating system or programming language" **[18]**. Web services are important for the client-server communication as they provide the solution for the interoperability problem. Web applications running on different platforms can communicate and exchange data using the same messaging format via web services.

There are different ways and languages to create a web service although the goal is always the same; the communication of the two sides over the same technology. Web services can be divided in the following three categories:

1. XML-RPC for messaging and XML schema to describe the service
2. SOAP messages and WSDL to describe the service
3. REST sending messages using the HTTP methods (GET, POST, PUT, DELETE)

Basically SOAP based web services are an enhanced approach of the XML-RPC method since both of them are using an XML format for messaging and an XML schema for describing their services. Thus, using the term **W**eb **S**ervices nowadays, we refer to SOAP web services. REST, on the other hand, is a more recent method of web service introduced to provide a simpler way of messaging than SOAP using just the four HTTP methods mentioned above.

### 3.3.1. SOAP Vs REST

In this chapter some examples of simple SOAP and REST based web services will be given, identifying, thus, their major characteristics and differences.

**SOAP**

Web Services based on SOAP messages are implemented using the following technologies:

1. SOAP (Simple Object Access Protocol)
2. WSDL (Web Services Description Language)
3. UDDI (Universal Description, Discovery and Integration)

SOAP is an XML-based language designed to allow messaging between different platforms over HTTP. It is a W3C recommendation and thus it is widely accepted and used in every operating system and browser providing interoperability between the web applications **[19]**.

WSDL is also a W3C recommendation and therefore can also be used in different platforms. It is written in the same style as an XML schema since it is used to describe a Web service. With WSDL we can define the location and the functionality of a web service.

UDDI is used to register web services designed for common purposes so that different applications can use the same service to cover their requirements. For example a web service designed for language translations can be adopted by several applications willing to include a Multilanguage property for their operations **[18]**. UDDI is an interesting technology although it is not always required for Web Service applications.

SOAP and WSDL technologies will be described in detail in the next chapters while UDDI is not required for the implementation of the application related to this project.

An example of soap Web Service request is shown below **[20]**.

```
GET /StockPrice HTTP/1.1
Host: example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: nnn

<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
   xmlns:s="http://www.example.org/stock-service">
   <env:Body>
     <s:GetStockQuote>
          <s:TickerSymbol>IBM</s:TickerSymbol>
     </s:GetStockQuote>
   </env:Body>
</env:Envelope>
```

**REST**

REST (Representational State Transfer) or RESTfull web services designed to give a much simpler approach for web services. They are called using one of the four HTTP

requests and the URL of the receiver. The supported HTTP request methods are GET, POST, UPDATE and DELETE. We are able to use SOAP messages and WSDL along with RESTfull web services but this is not required for their operation.

An example of REST web service request is shown below. Even from this small example we can see how simpler is a service request in REST web services **[21]**.

```
GET /StockPrice/IBM HTTP/1.1
Host: example.org
Accept: text/xml
Accept-Charset: utf-8
```

Comparing the two types of web services we can result that each of them may be more suitable for the implementation of different applications. Both of them are operating systems and platform independent. SOAP is more difficult to be learned and more complex in practise while REST is much simpler to be applied. SOAP can be implemented and described better as it uses additional technologies (i.e. WSDL) while REST is restricted to use only the four HTTP methods for communication. REST is only able to use HTTP protocol while SOAP can choose between additional protocols like SMTP and FTP. In general SOAP requires more processing time rather than REST **[21]**.

The communication model of the application related to this project has been implemented using REST web services. However SOAP and WSDL technologies are also included in this report since they have been considered during the design of the application.

### 3.3.2. SOAP

A SOAP message as described above is XML-based and its purpose is to exchange messages through Web Services. A SOAP message contains the following components:

- The Envelope element indicates that it is a SOAP message
- The Header element for header data
- The Body element used to include send and receive data
- The Fault element used to indicate possible errors.

The Envelope and Body elements are required while Header and Fault elements are optional. The syntax of a SOAP message is shown below. A real example of a SOAP message is included in the SOAP request included in the previous chapter.

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
...
</soap:Header>

<soap:Body>
...
  <soap:Fault>
  ...
  </soap:Fault>
</soap:Body>

</soap:Envelope>
```

### 3.3.3. WSDL

Web services description language is used to describe the message type, transport protocols and supported operations within a Web Service. Both sender and receiver must share the same WSDL for the same Web Service; the former to create the messages in the required format and the latter to process the received messages correctly and proceed with the appropriate actions **[22]**. A WSDL document contains the following sections:

- **types** of data involved in the Web Service.
- **message** exchanged in the Web service.
- **portType** to identify the operations expected in the Web Service.
- **binding** to describe the transport protocols of the Web Service.

The syntax of a WSDL document is shown below.

```
<definitions>

<types>
  definition of types........
</types>

<message>
  definition of a message....
</message>

<portType>
  definition of a port.......
</portType>

<binding>
  definition of a binding....
</binding>

</definitions>
```

Since WSDL is written in XML Schema style, types are declared using this format. Messages are declared as attributes used in the operations of a Web Service representing their Input and output parameters. The operations of a Web Service are defined inside the portType section. These operations can be either “one-way”, where there is no need for a response after the request or “two-way” if both request and response are required by the operation. An example of message and portType elements of a WSDL describing a glossary Web Service using a “request-response” operation is shown below **[23]**.

```
<message name="getTermRequest">
  <part name="term" type="xs:string"/>
</message>

<message name="getTermResponse">
  <part name="value" type="xs:string"/>
</message>

<portType name="glossaryTerms">
  <operation name="getTerm">
    <input message="getTermRequest"/>
    <output message="getTermResponse"/>
  </operation>
</portType>
```

The binding part defines the transport protocols and messaging types that are used for the Web Service we are describing. Binding takes two attributes. The first attribute gives the name of the binding and can take any value. The second attribute describes the type of the binding which must represent a portType element. If we want to use SOAP messages we should include a “soap:binding” element inside the binding part. “Soap:binding is also described using two attributes; the style and transport. Style attribute can take the values “rpc” or “document” only while transport defines the transport protocol which is going to be used such as HTTP or FTP. The binding element for the above example will be like the one shown below **[24]**.

```
<binding type="glossaryTerms" name="b1">
   <soap:binding style="document"
   transport="http://schemas.xmlsoap.org/soap/http" />
   <operation>
     <soap:operation soapAction="http://example.com/getTerm"/>
     <input><soap:body use="literal"/></input>
     <output><soap:body use="literal"/></output>
  </operation>
</binding>
```

## 3.4. Drawing simulations

For the purpose of this project, the designed web application is required to provide graphical representations illustrating the motion of the various dynamical systems which would be implemented. Moreover graphical representations must be able to be updated dynamically in order to describe the variation in the behaviour of the different systems after any possible modification on the initial attributes which affect the simulation.

Some of the dynamical systems such as the double pendulum may experience behaviours that are difficult to be clearly described using a two dimensional representation. For example in a double pendulum with its upper part swinging in a left-right pattern while its lower part swings forwards and backwards, would be very hard to realize both movements in a 2D representation. To overcome this, both 2D and 3D graphical representations have been implemented for the web application. Two dimensional graphics are being implemented using the canvas element of HTML. Hardware accelerated 3D graphics inside the browser can be generated with a new technology called WebGL. WebGL is based on OpenGL ES software library used for 3D rendering in many desktop and mobile applications.

### 3.4.1. Canvas

Canvas is an element of HTML 5.0. It is used for rendering images in a two dimensional space through JavaScript. Canvas is now being supported in the latest versions of all the common browsers. Firefox, Google Chrome and safari support canvas element from version 3.0, Opera form 10.0 while Internet explorer has recently included canvas support in its latest version 9.0. In addition canvas element is supported in the default browsers of both iPhone and android operating systems for smart-phones **[25]**. In this way, the designed Web application will be accessible from any mobile device running on these operating systems.

#### 3.4.1.1. Explorer canvas

Since Internet Explorer had very recently included canvas support, users of internet explorer, 7.0 and 8.0 are not able to access this functionality. To overcome this, Google development centre has designed a JavaScript library that translates canvas related commands into VML (Vector Markup Language). VML is a technology similar to SVG

(Scalable Vector Graphics) and it is supported in Internet explorer browsers since version 5.0. Through VML and this library, users of older versions of Internet Explorer can benefit the same functionality with other commonly used browsers [26].

The library is called explorer canvas and it is actually a script written in JavaScript so it is not required from users to install any additional plug-in for their browsers. The script functionality is implemented in a file called excanvas.js and it is widely available from Google development centre. To make web applications with canvas accessible from older versions of Internet Explorer, a programmer must include the script file in the application's source code in the same way as any other JavaScript file as shown in the code below.

```
1.  <script type="text/javascript" src="excanvas.js"></script>
```

#### 3.4.1.2. Canvas initialization

As mentioned above canvas element is being accessed and manipulated through JavaScript. However for JavaScript to be able to access canvas, an empty canvas element tag is required inside the HTML body. Canvas element will look like in the code below.

```
1.      <canvas id='canvasPanel' height='150' with='150'>
2.      This is an Html canvas element
3.      </canvas>
```

This will create a blank drawing panel surrounded by a regular border in black colour identifying the size of the panel as specified in the element declaration. We can add as many canvas elements as we wish including some extra information in order to specify the position of each element inside the web page. From the given code we can identify that canvas tag is not empty as mentioned above but it contains the sentence "This is an Html canvas element". This is not explicitly required for the use of the element and it will not be visible at any part inside or outside of the generated canvas panel. The reason that we may want to include some message inside the canvas element is in case that an older browser does not support canvas. In such case instead of the drawing panel the browser will display the message enclosed between the canvas tags. If browser does support canvas then the message will be ignored and a new canvas grid will be displayed inside the browser [25].

After creating the canvas panel, we can use its “id” attribute to access the element from the Document Object Model (DOM) and draw all the required shapes that we want to be displayed inside the panel. We can then reset the panel and re-draw the shapes in new positions visualising in this way the behaviour of a running dynamical system.

The following code illustrates how we can access the canvas element and start drawing different shapes.

```
1.   var canvas = document.getElementById('canvasPanel');
2.   var context = canvas.getContext('2d');
3.   context.fillRect(35,35,125,125);
```

The first line is used to create a reference to the canvas element defined in the DOM. Then we can access the rendering context of the element using the “getContext” method and passing a “2d” parameter indicating the rendering type we need to access. To access a 3D rendering context for canvas we need to use WebGL which is described in chapter 3.4.2. Context is where all the shapes are being drawn and then displayed inside the canvas panel [27].

### 3.4.1.3. Canvas shapes

The third line of the previous code is an example of the context’s usage to display a rectangle. There are three available methods to create rectangles; “FillRect” creates a rectangle with filled content, “StrokeRect” displays only the outline of the rectangle and “ClearRect” clears the canvas panel around and inside the rectangle shape. Each of these methods takes four attributes identifying the position and the size of the element. The first two attributes are the X and Y positions where the rectangle will be displayed inside the panel while the last two specify its width and height respectively. X and Y positions are calculated from the upper left corner of the canvas panel as shown in the figure below [28].

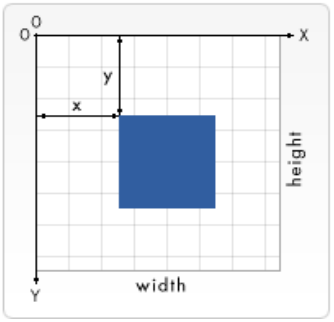



Figure 3.7: Canvas panel positions

Rectangles are the only shape that can be drawn directly using a single method of the context element. Any other shapes must be created with the use of different drawing paths. The code below will create a triangle using a set of lines.

```
1.      context.beginPath();       //start creating path
2.      context.moveTo(50,100);  //move to drawing position
3.      context.lineTo(100,50);    //create a line from current point to X,Y
4.      context.lineTo(10S0,100);  //create a line from current point to X,Y
5.      context.closePath();      //close path
6.      context.stroke();         //draw the outline of the shape
```

Before starting any path we must notify the context to receive the path using the method “beginPath()”. Afterwards, we use the “moveTo” command along with X and Y parameters to set the initial position of the path to be drawn. The “moveTo” command can be also used in-between a path to show that the next element of the path will start from a point different from the end point of the previous one. The “lineTo” command takes X and Y parameters, identifying the end point of the line to be drawn. “closePath” is used when we need to connect the last point with the starting point of the existing path to create a continuous shape. “Stroke” and “fill” have effects similar to the “strokeRect” and “fillRect” commands described above. “Stroke” creates the outline of the given path, while “fill” fills up the inside of the created shape. The “closePath” command is not required when we use the “fill” option since it is being used by default to make unclosed shapes able to be filled. The shape described by a new path is being displayed inside the canvas after the call of “fill” or “stroke” commands [28].

After the execution of the previous code, the triangle shape will be displayed inside the Canvas described below.

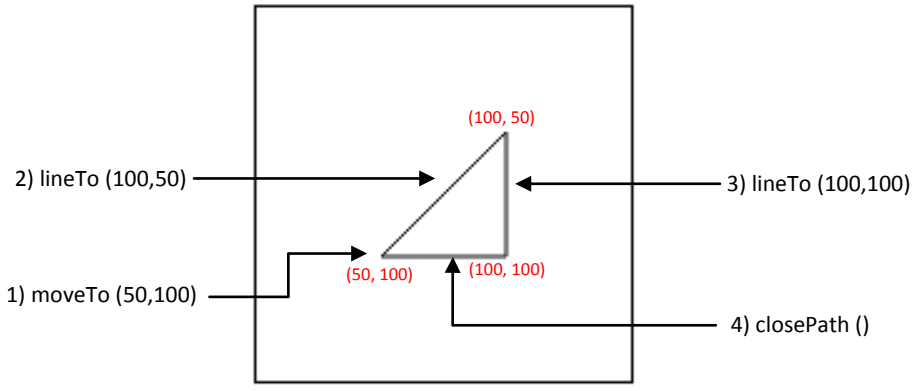



Figure 3.8: Canvas path example

Another interesting path type used in this project is the “Arc”, which allows the visualisation of different kinds of arcs and circles inside the canvas. For this project we use the Arc path is used to create the lower part of the pendulum simulations which is implemented in the shape of a ball. The Arc command takes six parameters order shown below.

```
1. context.arc(x, y, radius, initialAngle, finalAngle, anticlockwise)
```

X and Y denote the centre point of the arc. The third parameter sets the radius of the arc or the circle we want to create. Initial and final angles are counted in radians and are used to specify the start and end angles of the arc. To draw a circle, we can set the initial angle to 0 and the final angle to $2\pi$ or use any two angles which have a difference of $2\pi$ between them. The last attribute is a Boolean value identifying whether the arc will be drawn from the initial angle to the final angle in an anticlockwise (true) or a clockwise (false) direction [28]. The examples below describe the creation of two arcs which only differ in the anticlockwise parameter and one circle. It is also convenient to recognise which angle is treated by the arc command as 0. The result of each of the following commands is displayed in the figure below in same order.

```
1.      context.arc(10, 10, 3, 0, Math.PI, true);
2.      context.arc(10, 10, 3, 0, Math.PI, false);
3.      context.arc(10, 10, 3, 0, Math.PI*2, true);
```

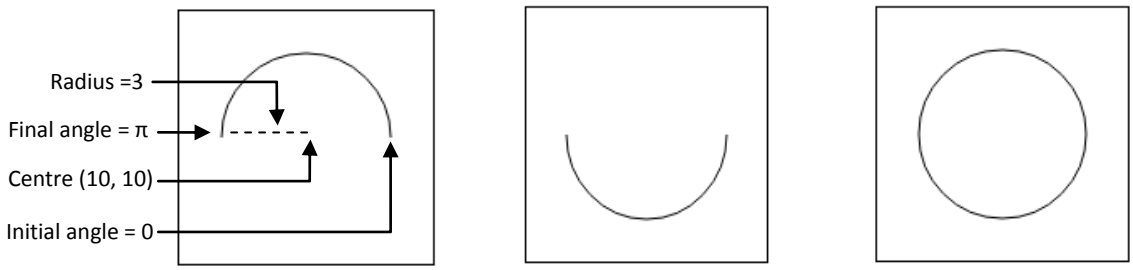



Figure 3.9: Canvas arc example

### 3.4.2. WebGL

As stated before WebGL (Web based Graphics Library) is a new technology that enables 3D graphics to be displayed inside the browser. Teams from both Mozilla and Google development centres were researching 3d rendering in web browsers quite

before the release of WebGL. Google was working on a cross-platform web API called "O3D" which could enable browsers to render 3D graphics after the installation of an additional software plug-in [29]. Mozilla's "canvas 3D" project was trying to enable 3D content to be displayed inside an HTML Canvas element [30]. Finally Khronos group took over what Mozilla started and designed the WebGL library [31]. Both "03D" and "canvas 3D" projects have then been transformed into WebGL implementations. Khronos group is the creator of OpenGL, OpenGL ES for non desktop devices (i.e. phones, consoles) and other similar technologies. "OpenGL® is the most widely adopted 2D and 3D graphics API in the industry" [32] while its ancestor OpenGL ES 2.0, as mentioned earlier is where WebGL functionality is based on.

WebGL is the only technology that made possible hardware accelerated 3D graphics to be used inside the browser without the need of any additional plug-ins. It is currently supported in the latest versions of the commonly used browsers except from Internet Explorer since Microsoft refuses to include WebGL support in Internet Explorer at the moment. This is not necessarily a negative sign if we consider that Internet Explorer had also included canvas support much later than the rest of the commonly used browsers. However, there is a plug-in available for Internet Explorer users who wish to experience WebGL functionality. WebGL is also available for mobile devices running on android operating systems via Firefox 4.0 browser for android.

The fact that WebGL is already supported by almost every commonly used browser and it does not require any plug-in installation, is the main reason that it has been selected for this project. WebGL is used to implement any dynamical systems included in the project which require a three dimensional space representation to describe their behaviour.

#### 3.4.2.1. Initialize WebGL

Similarly to Canvas described earlier, the rendering of the 3D information generated through WebGL is being displayed using the HTML Canvas element. The functionality of WebGL is implemented using JavaScript. The sample code that Mozilla provides in his tutorial on WebGL is based on the code of "Vlad Vukićević", which was involved in both canvas 3D project from Firefox and WebGL. In this code there are some basic functions created to separate the preparation of WebGL engine from the code used to generate the actual 3D objects. The same functions have been included in this project

for the initialisation of WebGL before drawing the scene. In the code below we can see the steps required to prepare and finally display the 3D graphics [33].

For this example we assume that the “Start_WebGL” function is called as soon as the web page loads using the “onLoad” property of HTML. However, it may be executed at any point of a program as soon as it follows the same order.

```
1.  function Start_WebGL() {
2.      var canvas = document.getElementById("canvas"); //get Canvas element
3.      initialize_GL(canvas);      //initialize the graphics library
4.      initialize_Shaders();      //initialize the shaders
5.      inititialize_Buffers();    //initialize buffers
6.      gl.clearColor(0.0, 0.0, 0.0, 1.0);    //clear the scene
7.      gl.enable(gl.DEPTH_TEST);             //enable depth test
8.      drawScene();                          //draw the scene
9.      }
```

Initially we need to locate the canvas element from the DOM using its id. Following we use the “initialize_GL” function which does a job similar to the Canvas initialization described in the previous chapter **3.4.2.1**. The only difference is that instead of passing the “2d” parameter in the “getContext()” method, it uses the word “WebGL” as rendering option. The output of the “getContext()” method is returned and stored in the “gl” variable which will be used in the subsequent functions [34].

Shaders is a fundamental part of WebGL since they used to translate the information describing the shape and coordinates of the different elements into actual screen pixels. Shaders are defined using a different language called GLSL. However we can include them inside the HTML code assigning an id attribute on each of them so we can then reference them through JavaScript. In every WebGL application we need to include two types of shaders, vertex-shader and fragment-shader. The vertex-shader calculates the position of the specified points for each shape along with any transformation or rotation operations used. In addition it calculates a lighting factor which will be used later in the fragment-shader. Fragment-shader is responsible to apply the required colours for each pixel, set its lighting using the lighting factor and display the pixel on the screen [35]. An example of Vertex and fragment shaders is available in appendix A.1.

### 3.4.2.2. Draw with WebGL

Buffer initialisation is where we define the points of the different shapes used to create a scene. As primitive shape, WebGL uses the triangle so if we want to create a rectangle we usually use a set of two identical orthogonal triangles. The same method applies even if we wish to create a sphere. Since WebGL does not have a read function to create a sphere, one way to achieve that is to use multiple triangles adjusted together along the latitude and longitude lines as shown in the figure below [36].

Figure 3.10: Sphere in WebGL

Buffers store the vertices of the shape we want to create in an array form. Vertices include every edge point of the shape holding three values for each point denoting its X, Y and Z positions [33]. For example a vertices array used to create a simple triangle will look like the one below.

```
1.        var triangle_vertices=[
2.           -1.0, 0.0, 0.0,      // X, Y, Z positions for first point
3.            1.0, 0.0. 0.0,      // X, Y, Z positions for second point
4.            0.0, 1.0, 0.0       // X, Y, Z positions for third point
5.            ];
```

The two methods called after the buffers' initialization are used to clear anything that has been previously added in the scene and enable depth testing to allow depth testing which means that objects closer to the viewing point will appear in front of others which have been placed behind them [33].

The last function is used to create the new scene. We can load and draw the objects defined in the buffer at any position in the scene using the transform or rotate methods to change their initial positions. The result is then displayed in the browser after calling the "gl.drawArrays()" method for every loaded buffer. "gl.drawArrays()"

method takes three attributes; the first one is the type of shapes included in the buffer (i.e. gl.TRIANGLES), the second one is the index of the first point to draw from the vertices array, and the third one is the number of points to draw starting from the first point [37]. If we want to create a 3D object we simply need to use more points inside the buffer vertices denoting the other parts of the shape. For example, to create a pyramid we will need four triangles similar to the one used in the code above, each of them with different point positions. To animate one or more objects we need to clear the scene and redraw the same objects in different positions during the period of time.

# 4. System functions

The designed system consists of four applications. The first application is called dimension reduction and is used to translate points given in a two dimensional space into one dimensional space. Users can add points in the two dimensional space (X, Y) using the provided graph while the result in the lower dimensional space is displayed in the form of a line passing through the origin of the graph. The angle of the line is calculated using the points added by the user. The input points for dimension reduction usually denote the output points of a dynamical system's run. The reason we apply dimension reduction is that computations are easier to be performed in lower dimensional spaces.

The second application describes the run of a single pendulum where users can edit the initial condition for the simulation like mass, length or gravity values. The simulation is illustrated using the canvas element described above. Along with the pendulum's simulation the application provides a graph plotter where user can select which values will be described in the graph. Graph plotter can display up to three colours of lines and each of them describes the values selected by the user.

The last two applications display the run of a double pendulum. The first of the two applications is using a 2D simulation based on the Canvas element described earlier while the second one uses the WebGL technology to display the simulation in three dimensions. The 3D version of the system provides a more understandable view of the double pendulum simulation since the user can change the viewing angle of the simulation and observe the system's behaviour from any position. Both 2D and 3D applications provide two graph plotters in the same form as for the single pendulum. The two plotters are used to describe the behaviour of the two pendulums (upper and lower) during the run of the simulation. As with the previous application, users are able to modify the initial condition of the dynamical system and the graph properties for each of the plotters.

The following section describes the available functionality in each of the four applications.

## 4.1. Use case diagram

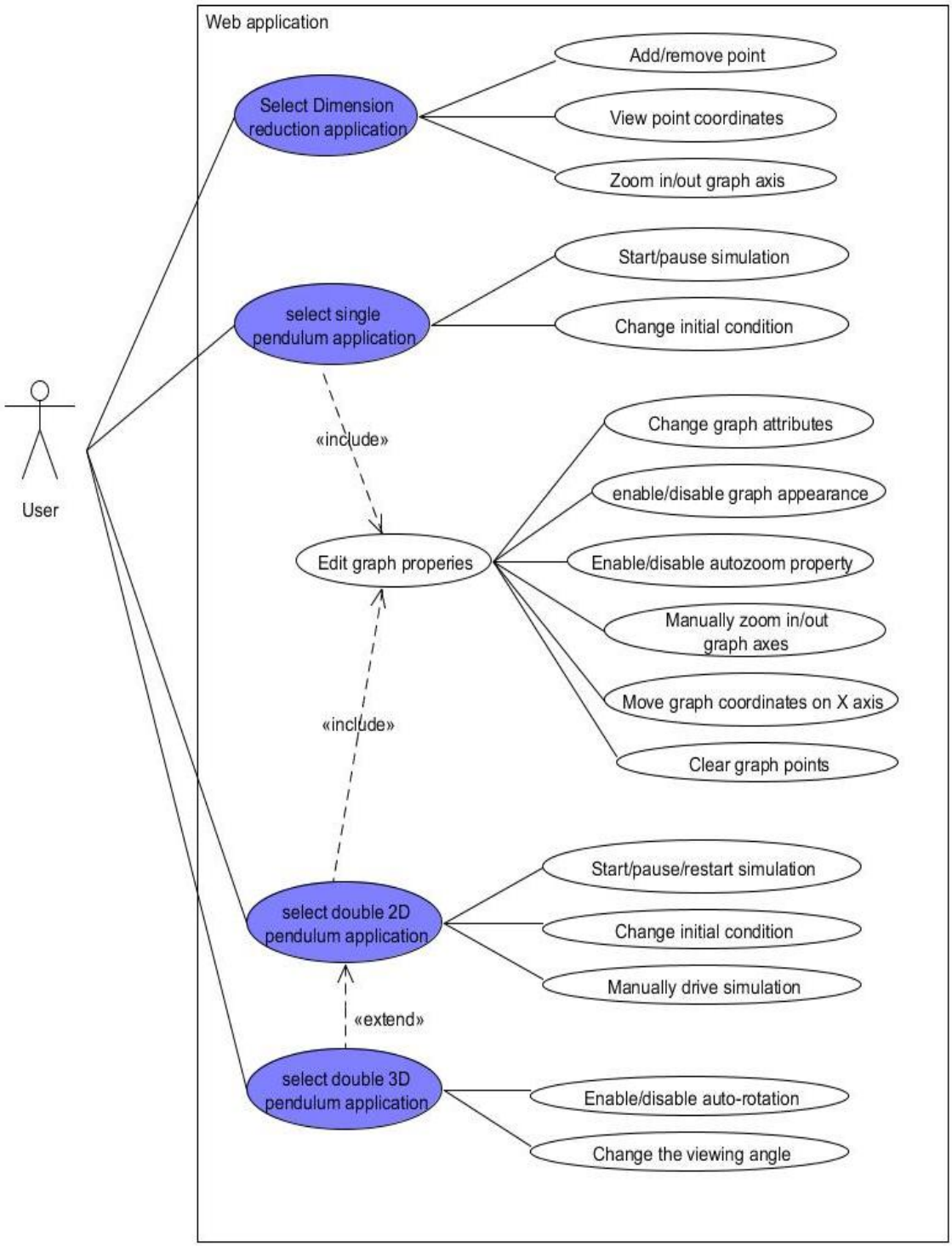



Figure 4.1: Use case diagram

## 4.2. Functions' description

### 4.2.1. "Select application" Use Case

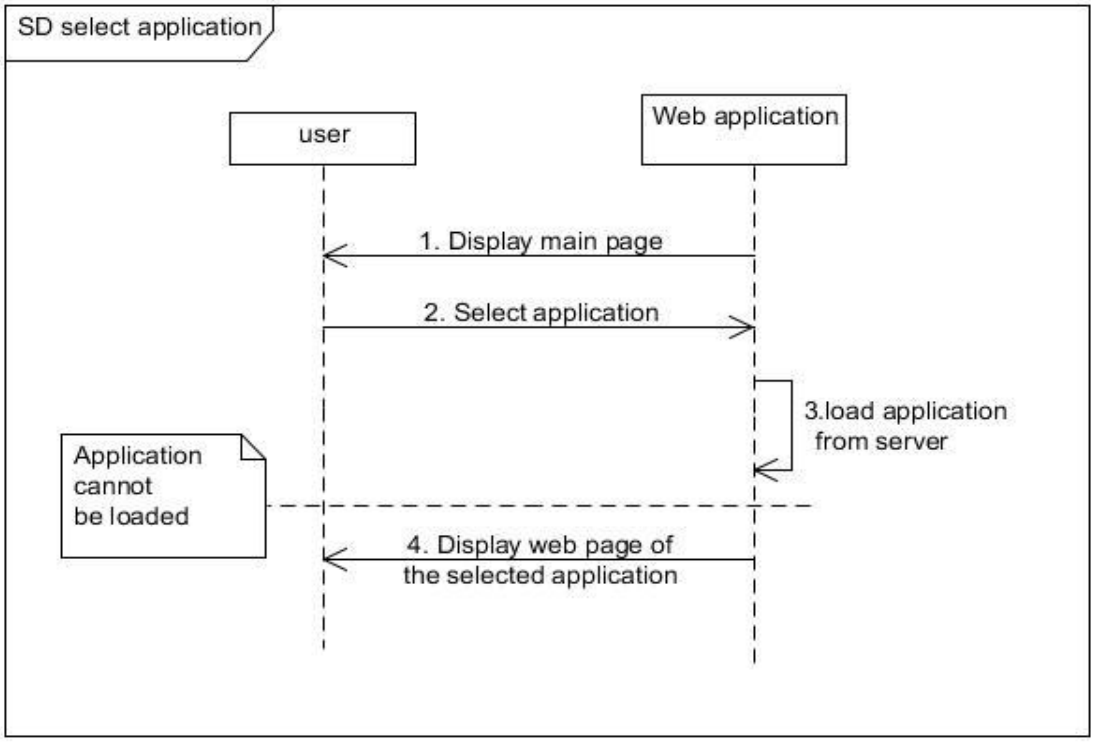
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Select application |
| ***Actor:*** | User |
| ***Goal:*** | Load the selected application in the browser |
| ***Precondition:*** | Main page of the system loaded |
| ***Postcondition:*** | Display the requested application's main page |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display web application's main page |
| 2 | User | Select application |
| 3 | Web application | Load application from server |
| 4 | Web application | Display web page of the selected application |

**Sequence Diagram**:




**Alternative Scenario:**

| Step | Alternative scenario condition | Response Description |
|---|---|---|
| 3 | Application cannot be loaded | Error message "cannot load application from remote server" – Go to step 1 |

### 4.2.2. "Dimension Reduction – add/remove point" Use Case

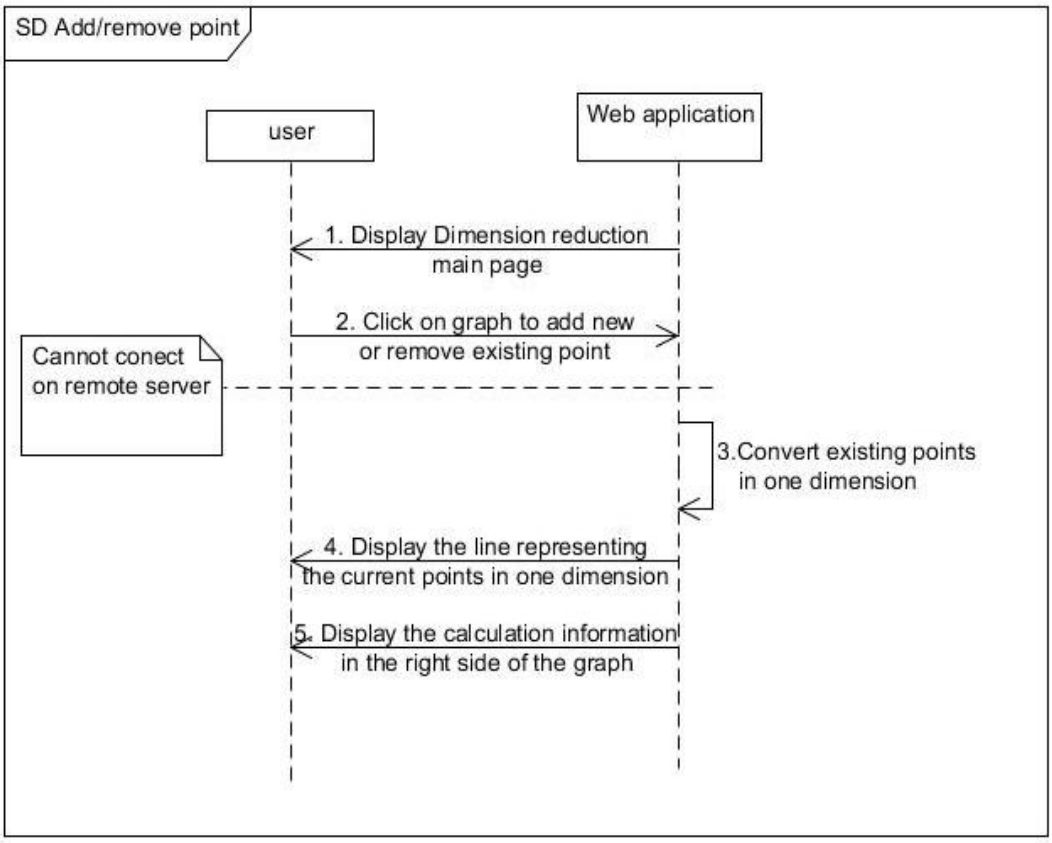
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Dimension Reduction – add/remove point |
| ***Actor:*** | User |
| ***Goal:*** | Add new point/remove existing point from graph |
| ***Precondition:*** | Main page of the Dimension Reduction application loaded |
| ***Postcondition:*** | Display a line representing the current points in one |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Dimension Reduction main page |
| 2 | User | Click on graph to add new or remove existing point |
| 3 | Web application | Convert existing points in one dimension |
| 4 | Web application | Display a line representing the current points in one dimension |
| 5 | Web application | Display the calculation information in the right side of the graph |

**Sequence Diagram**:


SD Add/remove point
user
Web application
1. Display Dimension reduction main page
2. Click on graph to add new or remove existing point
Cannot conect on remote server
3.Convert existing points in one dimension
4. Display the line representing the current points in one dimension
5. Display the calculation information in the right side of the graph

**Alternative Scenario:**

| Step | Alternative scenario condition | Response Description |
|---|---|---|
| 3 | Application cannot be loaded | Error message "cannot load application from remote server" – Go to step 1 |

### 4.2.3. "Dimension Reduction – view point coordinates" Use Case

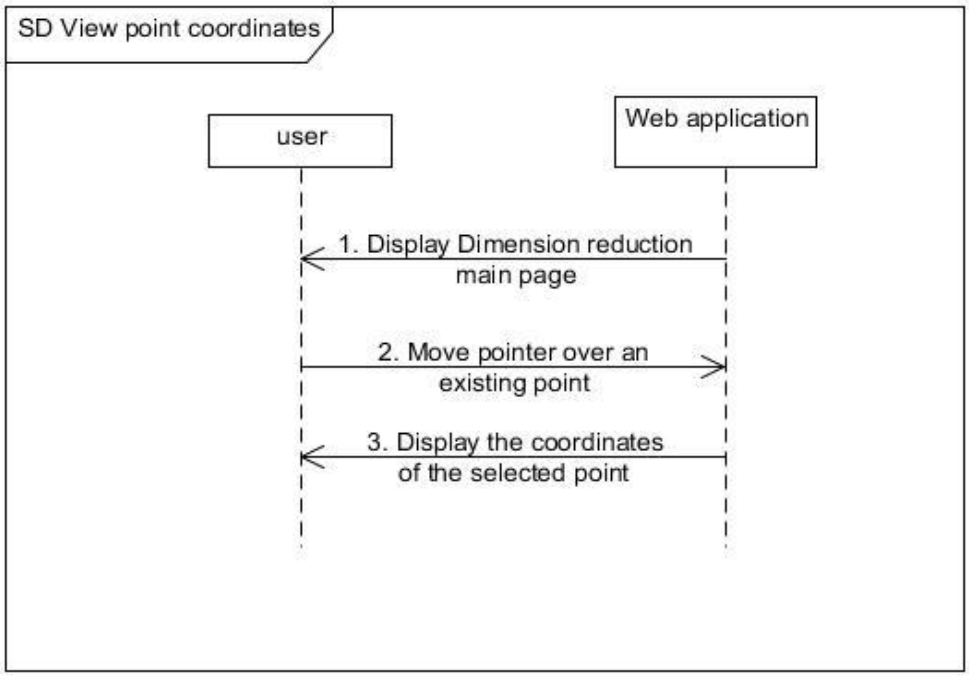
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Dimension Reduction – view point coordinates |
| ***Actor:*** | User |
| ***Goal:*** | Find the coordinates of the selected point |
| ***Precondition:*** | Main page of the Dimension Reduction application loaded |
| ***Postcondition:*** | Display the coordinates of the selected point |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Dimension Reduction main page |
| 2 | User | Move pointer over an existing point |
| 3 | Web application | Display the coordinates of the selected point |

**Sequence Diagram**:


SD View point coordinates
user
Web application
1. Display Dimension reduction main page
2. Move pointer over an existing point
3. Display the coordinates of the selected point

### 4.2.4. "Dimension Reduction – zoom in/out graph axis" Use Case

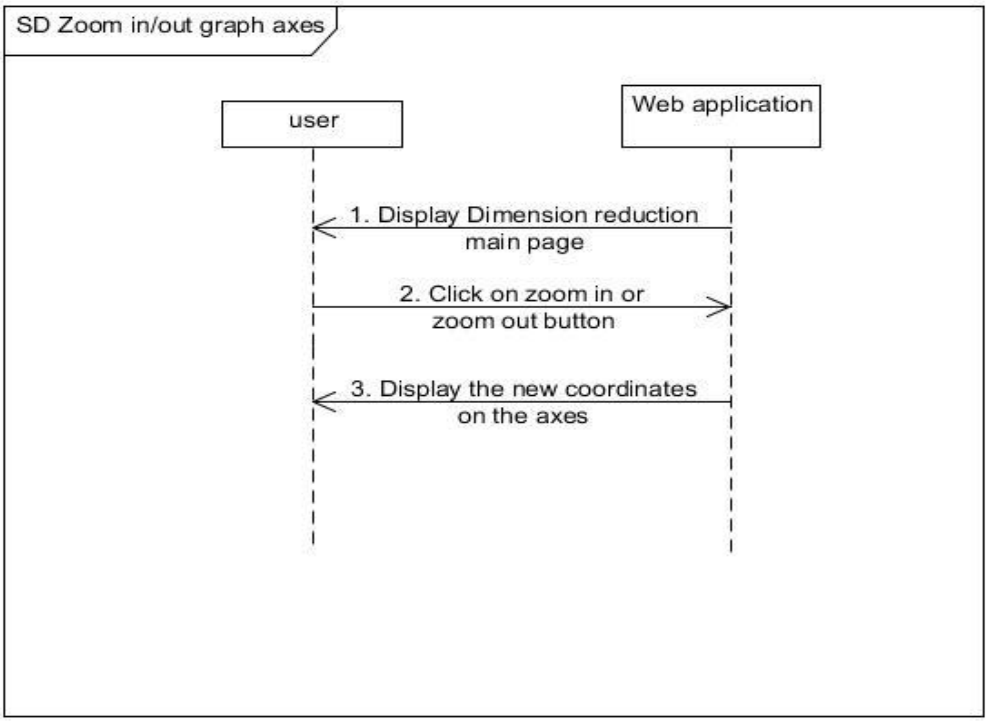
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Dimension Reduction – zoom in/out graph axis |
| ***Actor:*** | User |
| ***Goal:*** | Zoom in/out graph coordinates |
| ***Precondition:*** | Main page of the Dimension Reduction application loaded |
| ***Postcondition:*** | Display the new coordinates on each axis |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Dimension Reduction main page |
| 2 | User | Click on Zoom in/out button |
| 3 | Web application | Display the new coordinates on the axes |

**Sequence Diagram**:


SD Zoom in/out graph axes
user
Web application
1. Display Dimension reduction main page
2. Click on zoom in or zoom out button
3. Display the new coordinates on the axes

### 4.2.5. "Single pendulum – start/pause simulation" Use Case

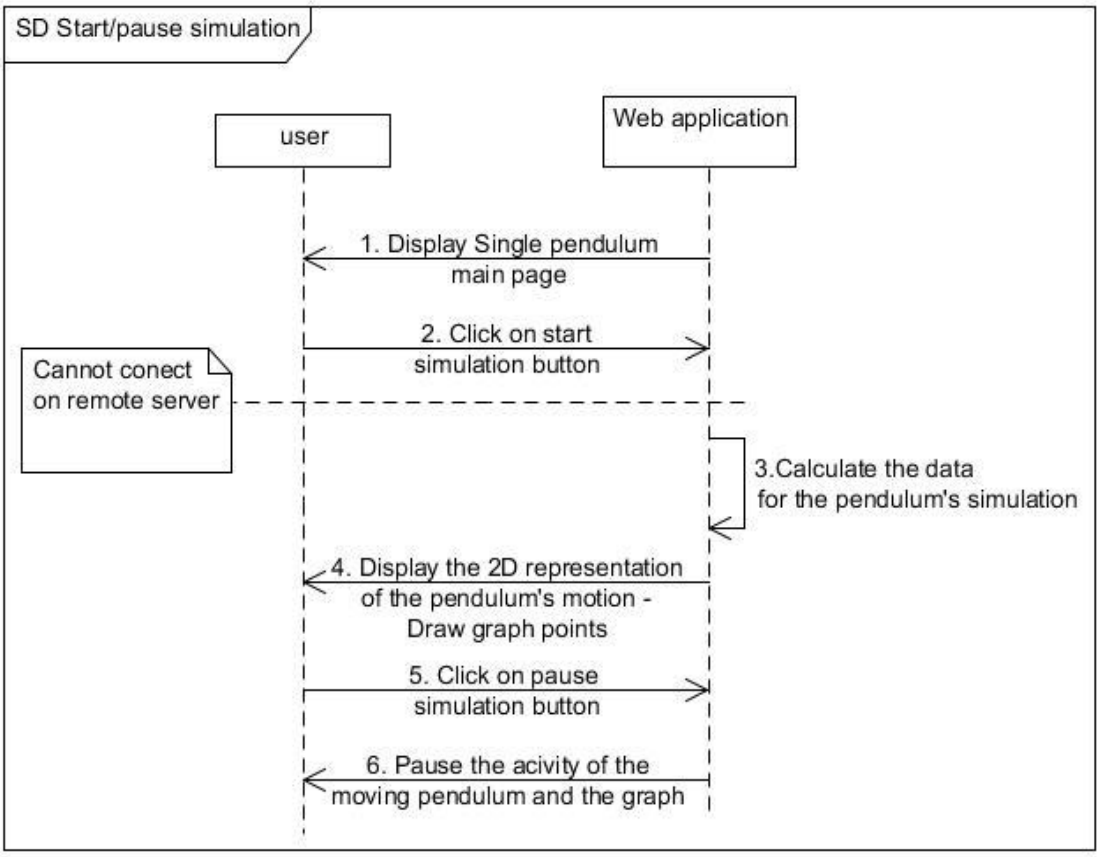
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Single pendulum – start/pause simulation |
| ***Actor:*** | User |
| ***Goal:*** | View single pendulum's simulation |
| ***Precondition:*** | Main page of the Single pendulum application loaded |
| ***Postcondition:*** | Pendulum simulation is running |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Single Pendulum main page |
| 2 | User | Click on start simulation button |
| 3 | Web application | Calculate the data for the pendulum simulation |
| 4 | Web application | Display the 2D representation of the pendulum's motion – Draw graph points |
| 5 | User | Click on pause simulation button |
| 6 | Web application | Pause the activity of the moving pendulum and the graph |

**Sequence Diagram**:


SD Start/pause simulation
user
Web application
1. Display Single pendulum main page
2. Click on start simulation button
Cannot conect on remote server
3.Calculate the data for the pendulum's simulation
4. Display the 2D representation of the pendulum's motion - Draw graph points
5. Click on pause simulation button
6. Pause the acivity of the moving pendulum and the graph

**Alternative Scenario:**

| Step | Alternative scenario condition | Response Description |
|---|---|---|
| 3 | Cannot connect on remote server | Error message "cannot connect on remote server" – Go to step 1 |

### 4.2.6. "Single pendulum – change initial condition" Use Case

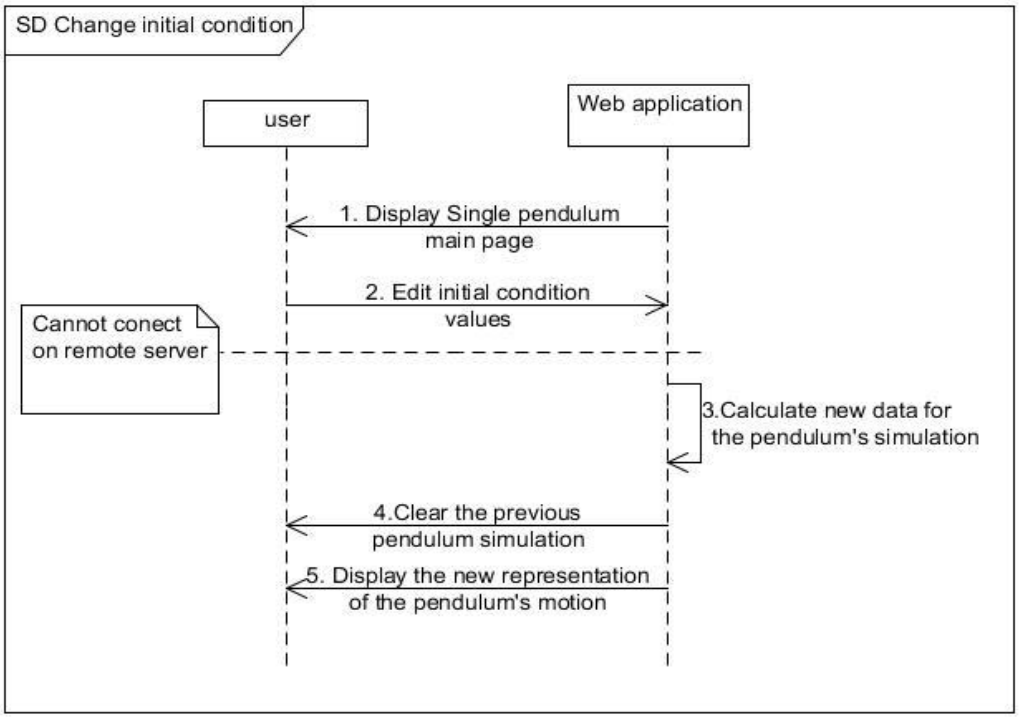
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Single pendulum – change initial condition |
| ***Actor:*** | User |
| ***Goal:*** | View the pendulum's simulation based on the new initial condition |
| ***Precondition:*** | Main page of the Single pendulum application loaded |
| ***Postcondition:*** | The updated pendulum simulation is running |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Single Pendulum main page |
| 2 | User | Edit initial condition values |
| 3 | Web application | Calculate new data for the pendulum simulation |
| 4 | Web application | Clear the previous the pendulum simulation |
| 5 | Web application | Display the new representation of the pendulum's motion |

**Sequence Diagram**:


SD Change initial condition
user
Web application
1. Display Single pendulum main page
2. Edit initial condition values
Cannot conect on remote server
3.Calculate new data for the pendulum's simulation
4.Clear the previous pendulum simulation
5. Display the new representation of the pendulum's motion

**Alternative Scenario:**

| Step | Alternative scenario condition | Response Description |
|---|---|---|
| 3 | Cannot connect on remote server | Error message "cannot connect on remote server" – Go to step 1 |

### 4.2.7. "Double pendulum – start/pause/restart simulation" Use Case

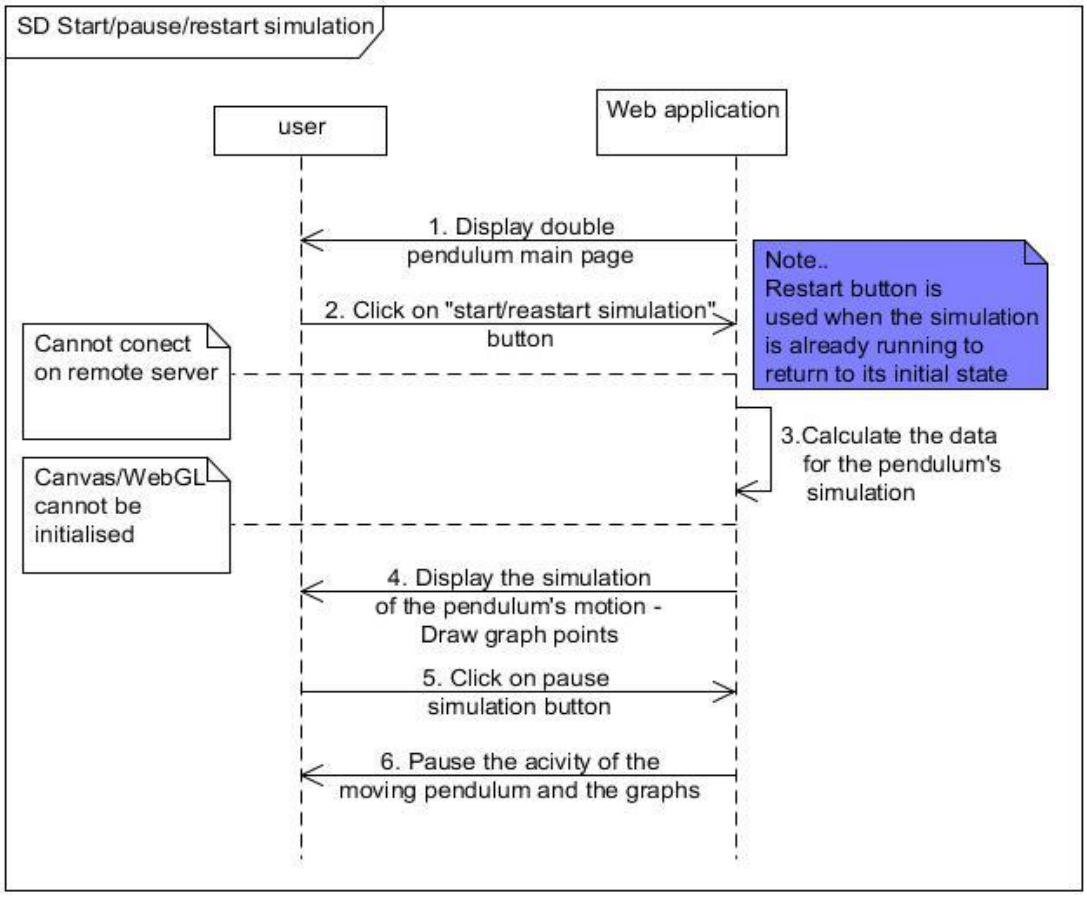
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Double pendulum – start/pause/restart simulation |
| ***Actor:*** | User |
| ***Goal:*** | View Double pendulum's simulation |
| ***Precondition:*** | Main page of the Double pendulum application loaded |
| ***Postcondition:*** | Double Pendulum simulation is running |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Double Pendulum main page |
| 2 | User | Click on start/restart simulation button |
| 3 | Web application | Calculate the data for the pendulum simulation |
| 4 | Web application | Display the simulation of the pendulum's motion – Draw graph points |
| 5 | User | Click on pause simulation button |
| 6 | Web application | Pause the activity of the moving pendulum and the graphs |

**Sequence Diagram**:


SD Start/pause/restart simulation
user
Web application
1. Display double pendulum main page
2. Click on "start/reastart simulation" button
Note..
Restart button is used when the simulation is already running to return to its initial state
Cannot conect on remote server
3.Calculate the data for the pendulum's simulation
Canvas/WebGL cannot be initialised
4. Display the simulation of the pendulum's motion - Draw graph points
5. Click on pause simulation button
6. Pause the acivity of the moving pendulum and the graphs

**Alternative Scenario:**

| Step | Alternative scenario condition | Response Description |
|---|---|---|
| 3 | Cannot connect on remote server | Error message "cannot connect on remote server" – Go to step 1 |
| 4 | Canvas/WebGL cannot be initialised | Error message "canvas/WebGL cannot be initialised" – Draw graph simulation only |

### 4.2.8. "Double pendulum – change initial condition" Use Case

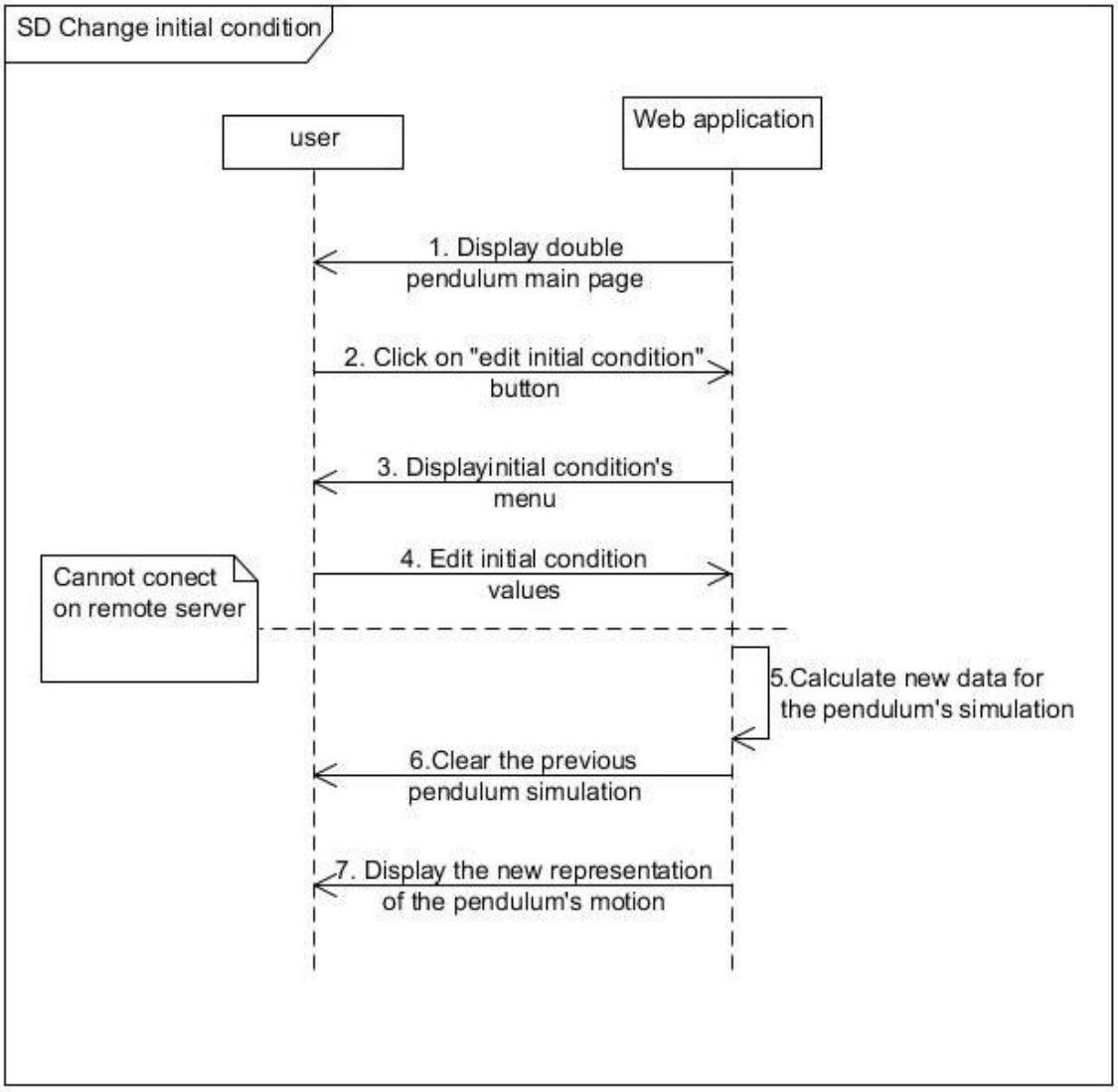
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Double pendulum – change initial condition |
| ***Actor:*** | User |
| ***Goal:*** | View pendulum simulation based on the new initial condition |
| ***Precondition:*** | Main page of the Double pendulum application loaded |
| ***Postcondition:*** | The updated pendulum simulation is running |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Double Pendulum main page |
| 2 | User | Click on "edit initial condition" button |
| 3 | Web application | Display initial condition's menu |
| 4 | Web application | Edit initial condition values |
| 5 | Web application | Calculate new data for the pendulum simulation |
| 6 | Web application | Clear the previous the pendulum simulation |
| 7 | Web application | Display the new representation of the pendulum's motion |

**Sequence Diagram**:


SD Change initial condition
user
Web application
1. Display double pendulum main page
2. Click on "edit initial condition" button
3. Displayinitial condition's menu
4. Edit initial condition values
Cannot conect on remote server
5.Calculate new data for the pendulum's simulation
6.Clear the previous pendulum simulation
7. Display the new representation of the pendulum's motion

**Alternative Scenario:**

| Step | Alternative scenario condition | Response Description |
|---|---|---|
| 3 | Cannot connect on remote server | Error message "cannot connect on remote server" – Go to step 1 |

### 4.2.9. "Double pendulum – Manually drive simulation" Use Case

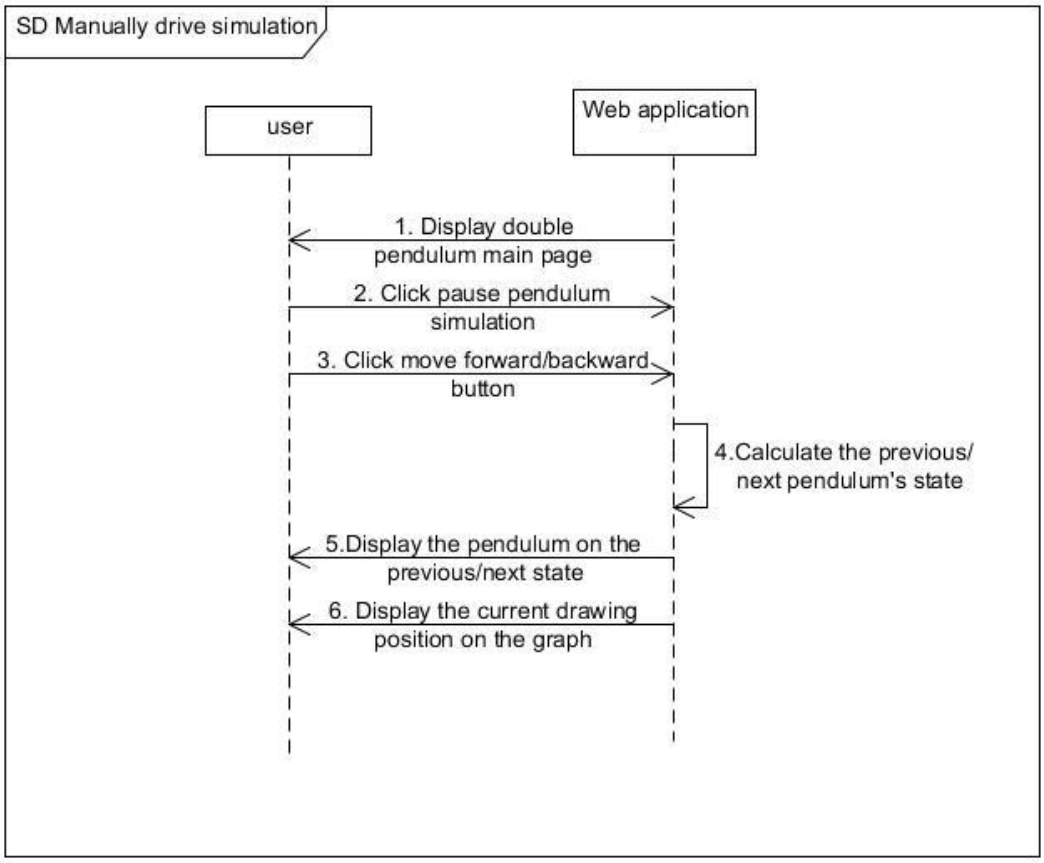
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Double pendulum – manually drive simulation |
| ***Actor:*** | User |
| ***Goal:*** | View next or previous state of the simulation |
| ***Precondition:*** | Double pendulum simulation is paused |
| ***Postcondition:*** | Double Pendulum simulation move on the next/previous |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Double Pendulum main page |
| 2 | User | Click pause pendulum simulation |
| 3 | User | Click move forward/backward button |
| 4 | Web application | Calculate the previous/next pendulum's state |
| 5 | Web application | Display the pendulum on the previous/next state |
| 6 | Web application | Display the current drawing position on the graph |

**Sequence Diagram**:


SD Manually drive simulation
user
Web application
1. Display double pendulum main page
2. Click pause pendulum simulation
3. Click move forward/backward button
4.Calculate the previous/ next pendulum's state
5.Display the pendulum on the previous/next state
6. Display the current drawing position on the graph

### 4.2.10. "Double pendulum 3D – enable/disable autorotation" Use Case

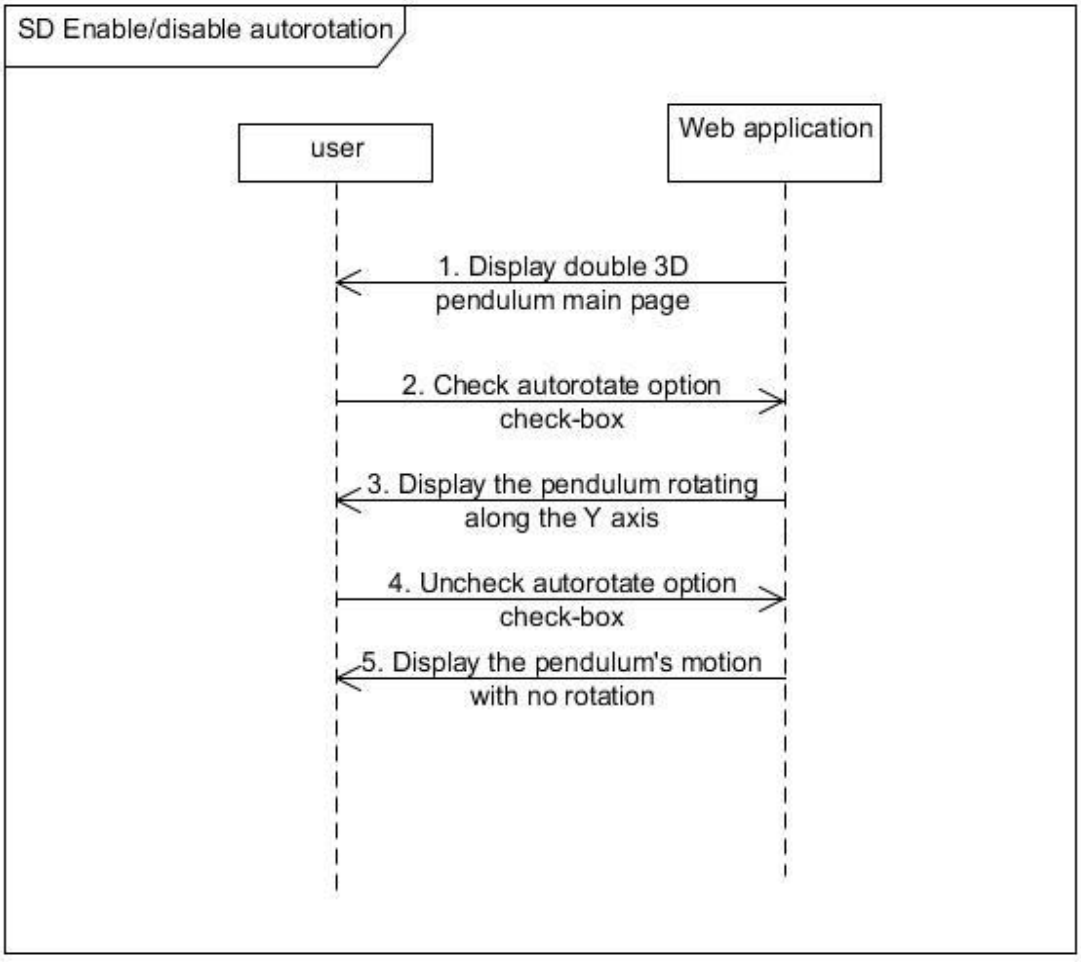
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Double pendulum 3D – enable/disable autorotation |
| ***Actor:*** | User |
| ***Goal:*** | Display the pendulum rotating along Y axis during its run |
| ***Precondition:*** | Double pendulum 3D application loaded |
| ***Postcondition:*** | Double Pendulum simulation rotating along the Y axis |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Double 3D Pendulum main page |
| 2 | User | Check autorotate option check-box |
| 3 | Web application | Display the pendulum rotating along the Y axis |
| 4 | User | Uncheck autorotate option check-box |
| 5 | Web application | Display the pendulum's motion with no rotation |

**Sequence Diagram**:


SD Enable/disable autorotation
user
Web application
1. Display double 3D pendulum main page
2. Check autorotate option check-box
3. Display the pendulum rotating along the Y axis
4. Uncheck autorotate option check-box
5. Display the pendulum's motion with no rotation

### 4.2.11. "Double pendulum 3D – Change the viewing angle" Use Case

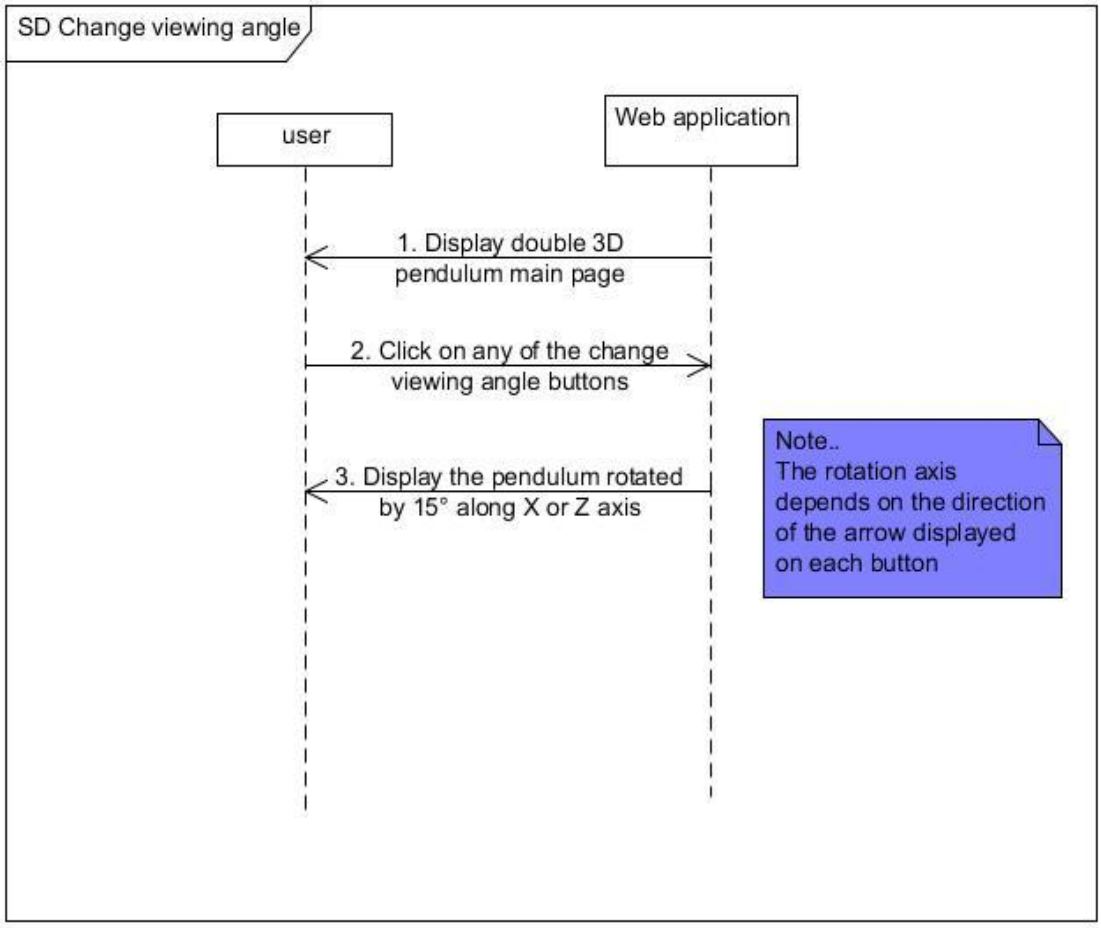
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Double pendulum 3D – Change the viewing angle |
| ***Actor:*** | User |
| ***Goal:*** | Display the pendulum rotated along Y and/or Z axes |
| ***Precondition:*** | Double pendulum 3D application loaded |
| ***Postcondition:*** | Double Pendulum simulation rotated by 15° in the requested direction |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display Double 3D Pendulum main page |
| 2 | User | Click on any of the "change viewing angle" buttons |
| 3 | Web application | Display the pendulum rotated by 15 along |

**Sequence Diagram**:


SD Change viewing angle
user
Web application
1. Display double 3D pendulum main page
2. Click on any of the change viewing angle buttons
3. Display the pendulum rotated by 15° along X or Z axis
Note..
The rotation axis depends on the direction of the arrow displayed on each button

### 4.2.12. "Edit graph properties – Change graph attributes" Use Case

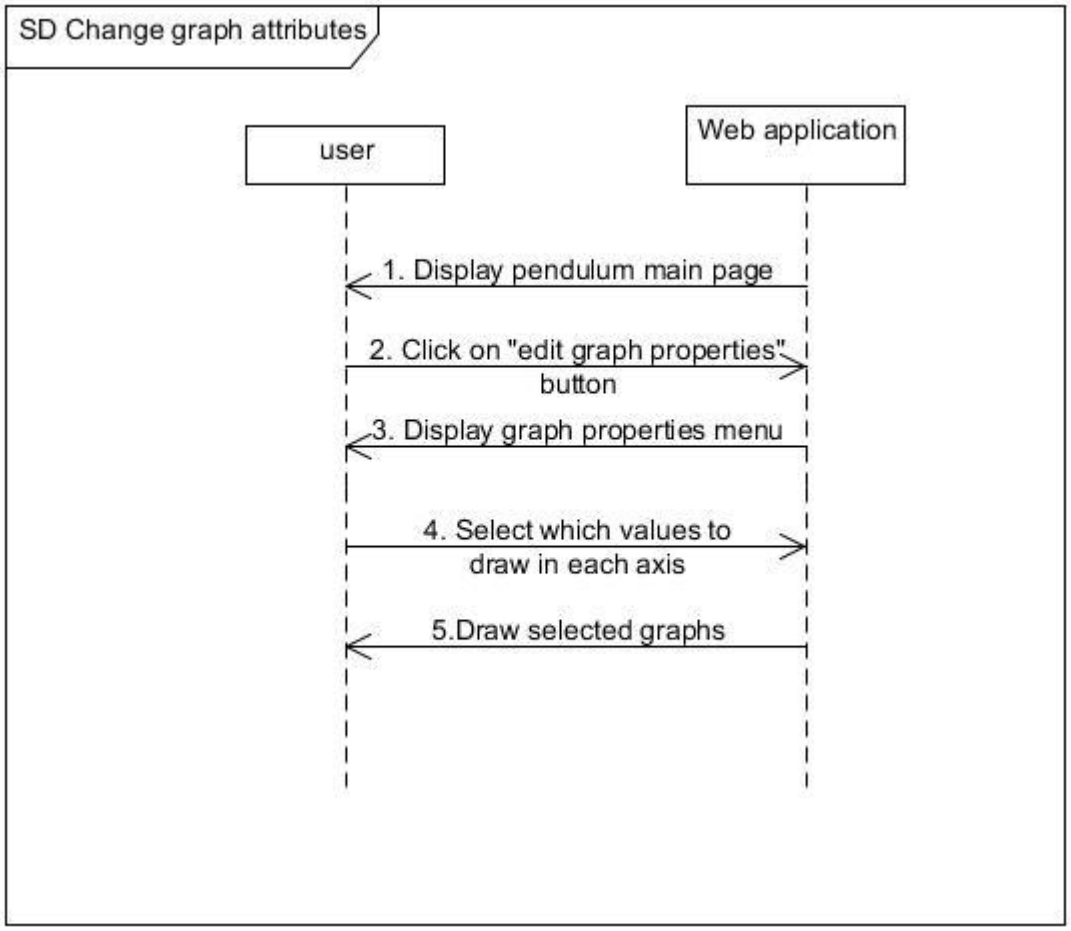
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Edit graph properties – change graph attributes |
| ***Actor:*** | User |
| ***Goal:*** | Change the graph values described on each axis |
| ***Precondition:*** | Pendulum application loaded |
| ***Postcondition:*** | Draw the new graph values in the plotter |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display pendulum main page |
| 2 | User | Click on "edit graph properties" button |
| 3 | Web application | Display graph properties menu |
| 4 | User | Select which values to draw in each axis |
| 5 | Web application | Draw selected graphs |

**Sequence Diagram**:


SD Change graph attributes
user
Web application
1. Display pendulum main page
2. Click on "edit graph properties" button
3. Display graph properties menu
4. Select which values to draw in each axis
5.Draw selected graphs

### 4.2.13. "Edit graph properties – Change graph appearance" Use Case

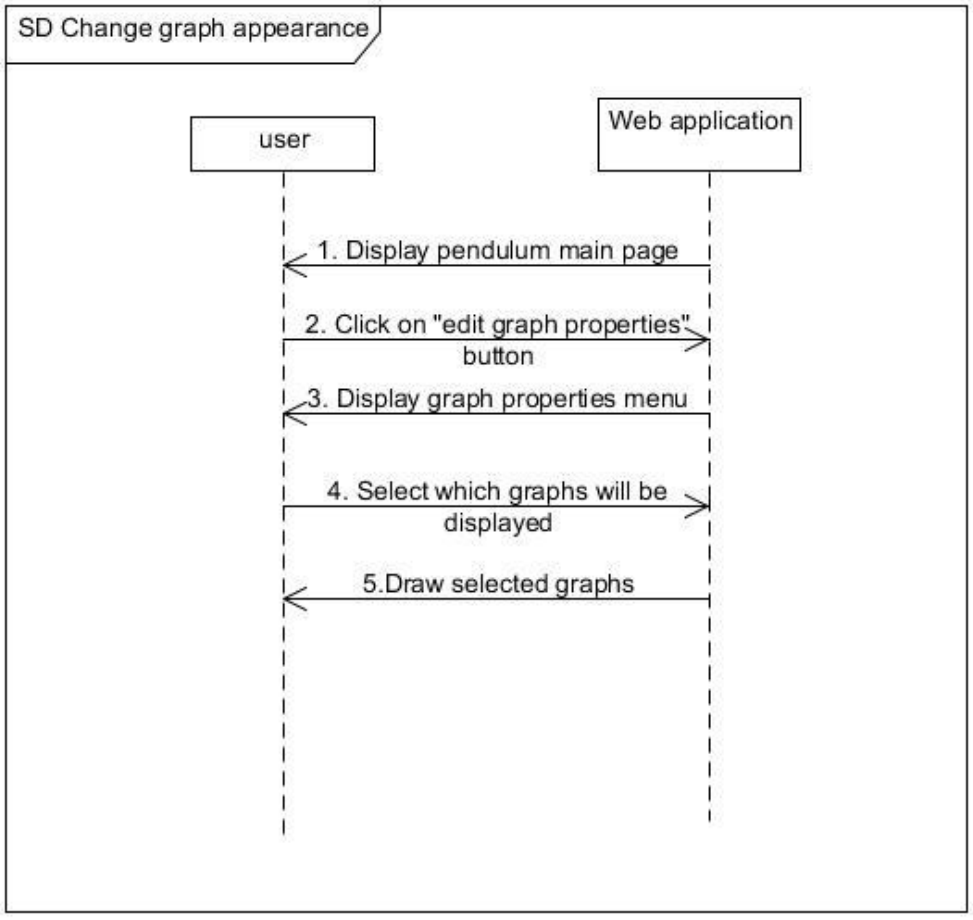
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Edit graph properties – change graph appearance |
| ***Actor:*** | User |
| ***Goal:*** | Edit the number of graphs displayed on each plotter |
| ***Precondition:*** | Pendulum application loaded |
| ***Postcondition:*** | The selected graphs appear in the plotter |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display pendulum main page |
| 2 | User | Click on "edit graph properties" button |
| 3 | Web application | Display graph properties menu |
| 4 | User | Select which graphs will be displayed |
| 5 | Web application | Draw selected graphs |

**Sequence Diagram**:


SD Change graph appearance
user
Web application
1. Display pendulum main page
2. Click on "edit graph properties" button
3. Display graph properties menu
4. Select which graphs will be displayed
5.Draw selected graphs

### 4.2.14. "Edit graph properties – enable/disable autozoom property" Use Case

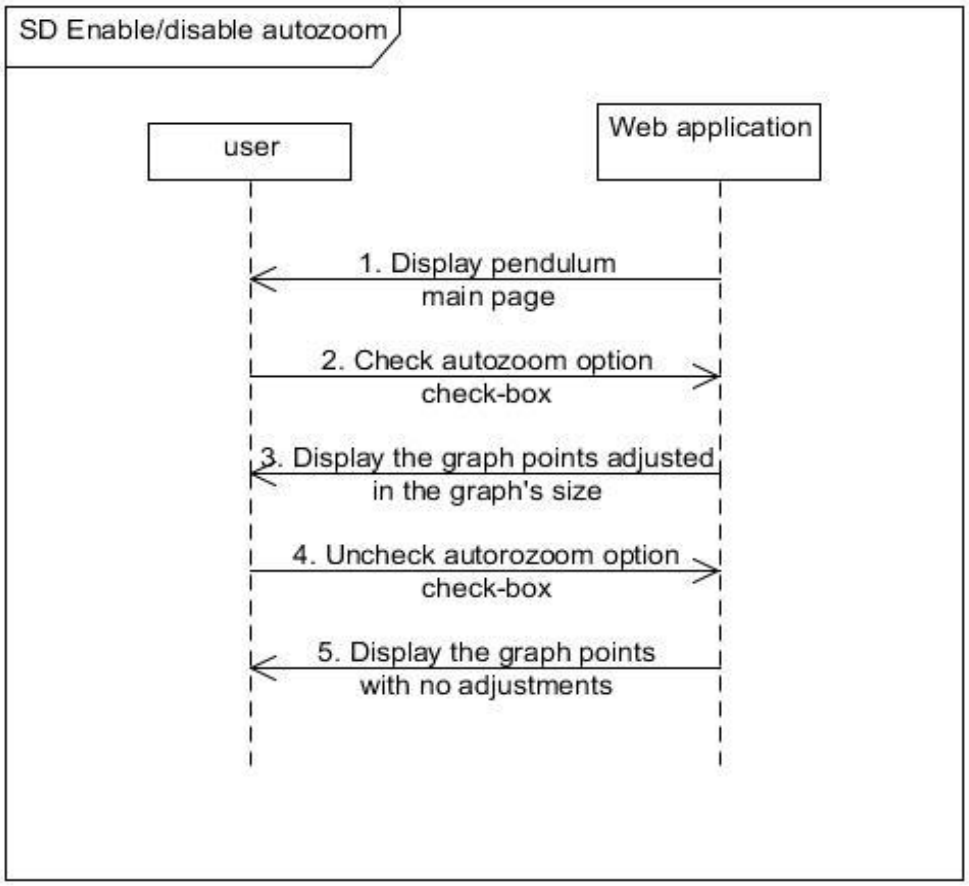
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Edit graph properties – enable/disable autozoom property |
| ***Actor:*** | User |
| ***Goal:*** | Graph points are zoomed in/out to fit the plotter |
| ***Precondition:*** | Pendulum application loaded |
| ***Postcondition:*** | Graph points adjusted to fit the graph size |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display pendulum main page |
| 2 | User | Check autozoom option check-box |
| 3 | Web application | Display graph points adjusted in the graph's size |
| 4 | User | Uncheck autozoom option check-box |
| 5 | Web application | Display he graph point with no adjustments |

**Sequence Diagram**:


SD Enable/disable autozoom
user
Web application
1. Display pendulum main page
2. Check autozoom option check-box
3. Display the graph points adjusted in the graph's size
4. Uncheck autorozoom option check-box
5. Display the graph points with no adjustments

### 4.2.15. "Edit graph properties – manually zoom in/out graph axes" Use Case

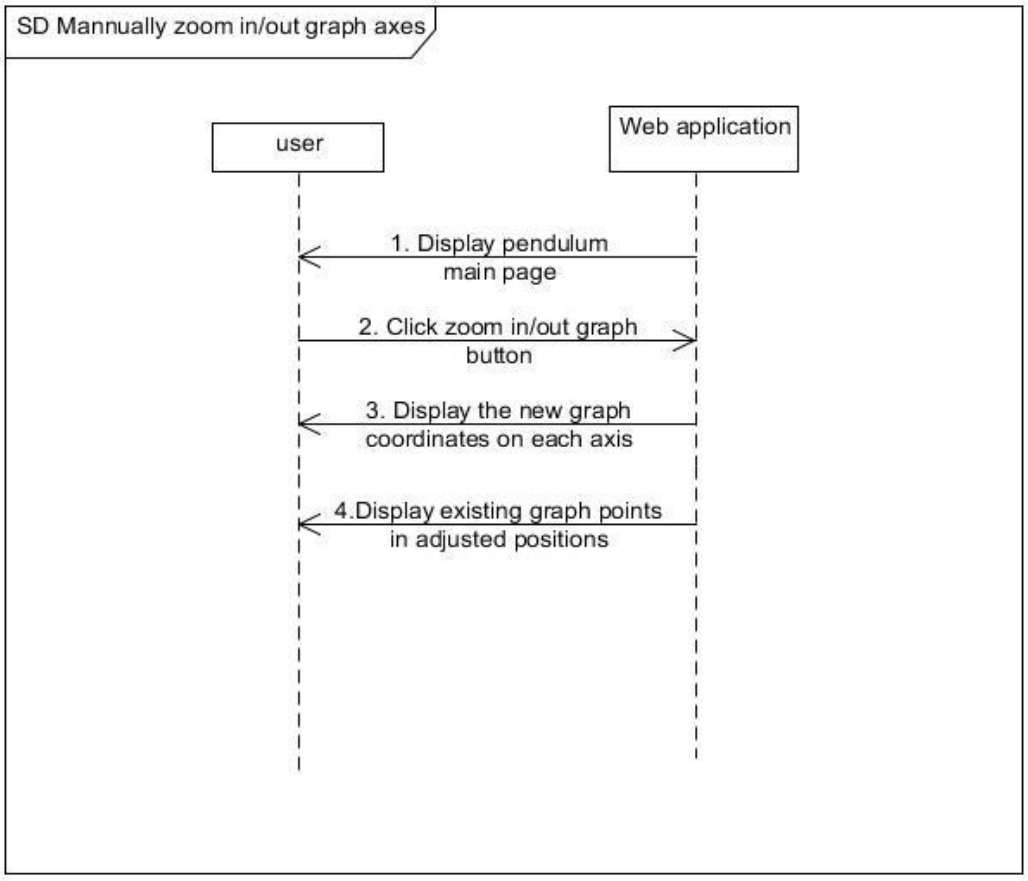
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Edit graph properties – manually zoom in/out graph axes |
| ***Actor:*** | User |
| ***Goal:*** | Zoom in/out graph points in the required axis |
| ***Precondition:*** | Pendulum application loaded |
| ***Postcondition:*** | Display graph points zoomed in/out as requested |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display pendulum main page |
| 2 | User | Click zoom in/out graph button |
| 3 | Web application | Display the new graph coordinates on each axis |
| 4 | Web application | Display existing graph points in adjusted positions |

**Sequence Diagram**:


SD Mannually zoom in/out graph axes
user
Web application
1. Display pendulum main page
2. Click zoom in/out graph button
3. Display the new graph coordinates on each axis
4.Display existing graph points in adjusted positions

### 4.2.16. "Edit graph properties – move graph coordinates on X axis" Use Case

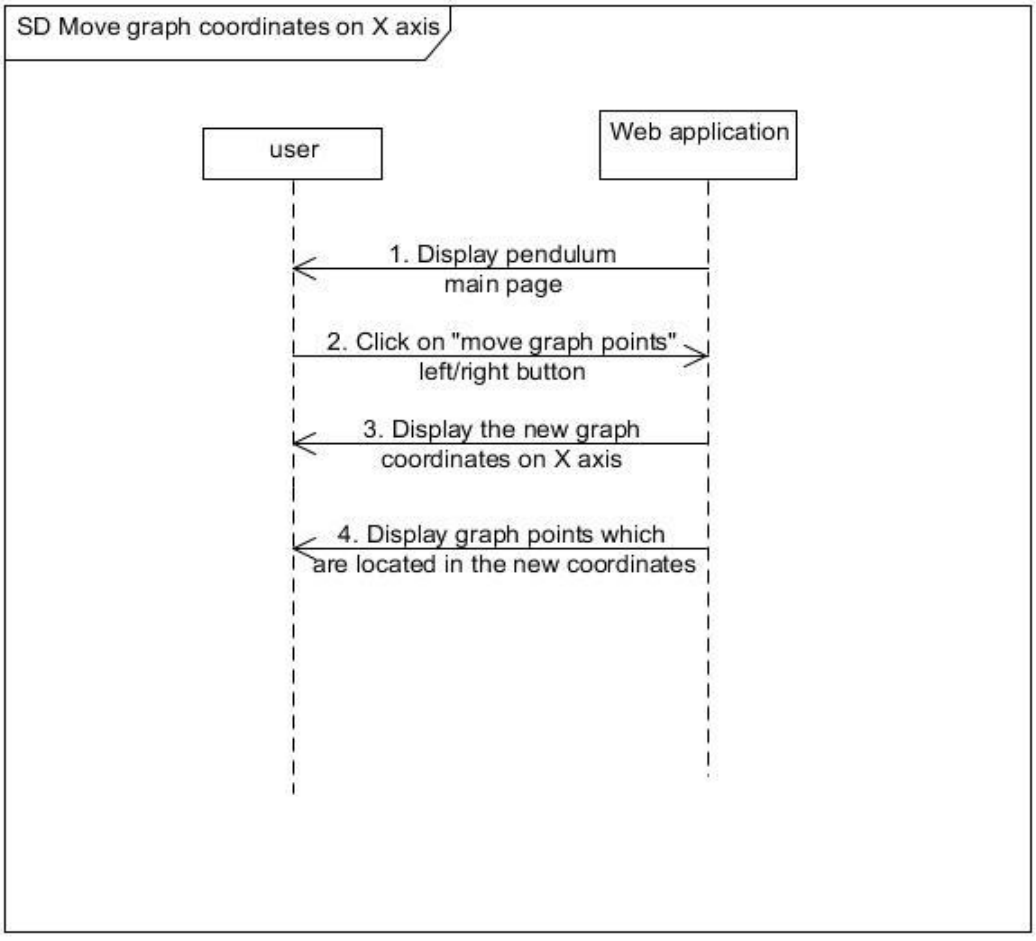
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Edit graph properties – move graph coordinates on X axis |
| ***Actor:*** | User |
| ***Goal:*** | Shift the X coordinates in left or right direction and display the current graph points for the new coordinates |
| ***Precondition:*** | Pendulum application loaded |
| ***Postcondition:*** | Display coordinates and points shifted in the requested direction |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display pendulum main page |
| 2 | User | Click on "move graph points" left/right |
| 3 | Web application | Display the new graph coordinates on X axis |
| 4 | Web application | Display graph points which are located in the new coordinates |

**Sequence Diagram**:


SD Move graph coordinates on X axis
user
Web application
1. Display pendulum main page
2. Click on "move graph points" left/right button
3. Display the new graph coordinates on X axis
4. Display graph points which are located in the new coordinates

### 4.2.17. "Edit graph properties – clear graph" Use Case

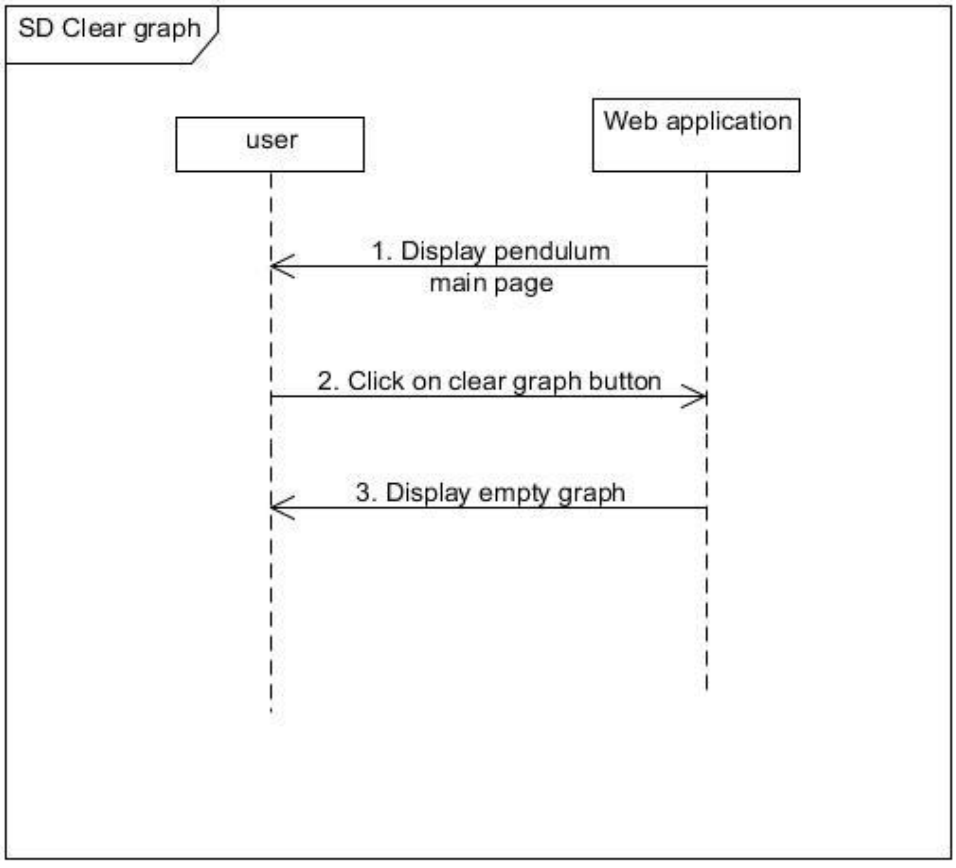
**Use Case properties:**

| | |
|---|---|
| ***Name:*** | Edit graph properties – clear graph |
| ***Actor:*** | User |
| ***Goal:*** | Clear all the current points of the graph |
| ***Precondition:*** | Pendulum application loaded |
| ***Postcondition:*** | Display empty graph |
| ***System:*** | Virtual Executive Toys Web Application |

**Description:**

| Step | Actor/System | Activity Description |
|---|---|---|
| 1 | Web application | Display pendulum main page |
| 2 | User | Click on clear graph button |
| 3 | Web application | Display empty graph |

**Sequence Diagram**:



# 5. System and Technology Architecture

## 5.1. Model of the problem

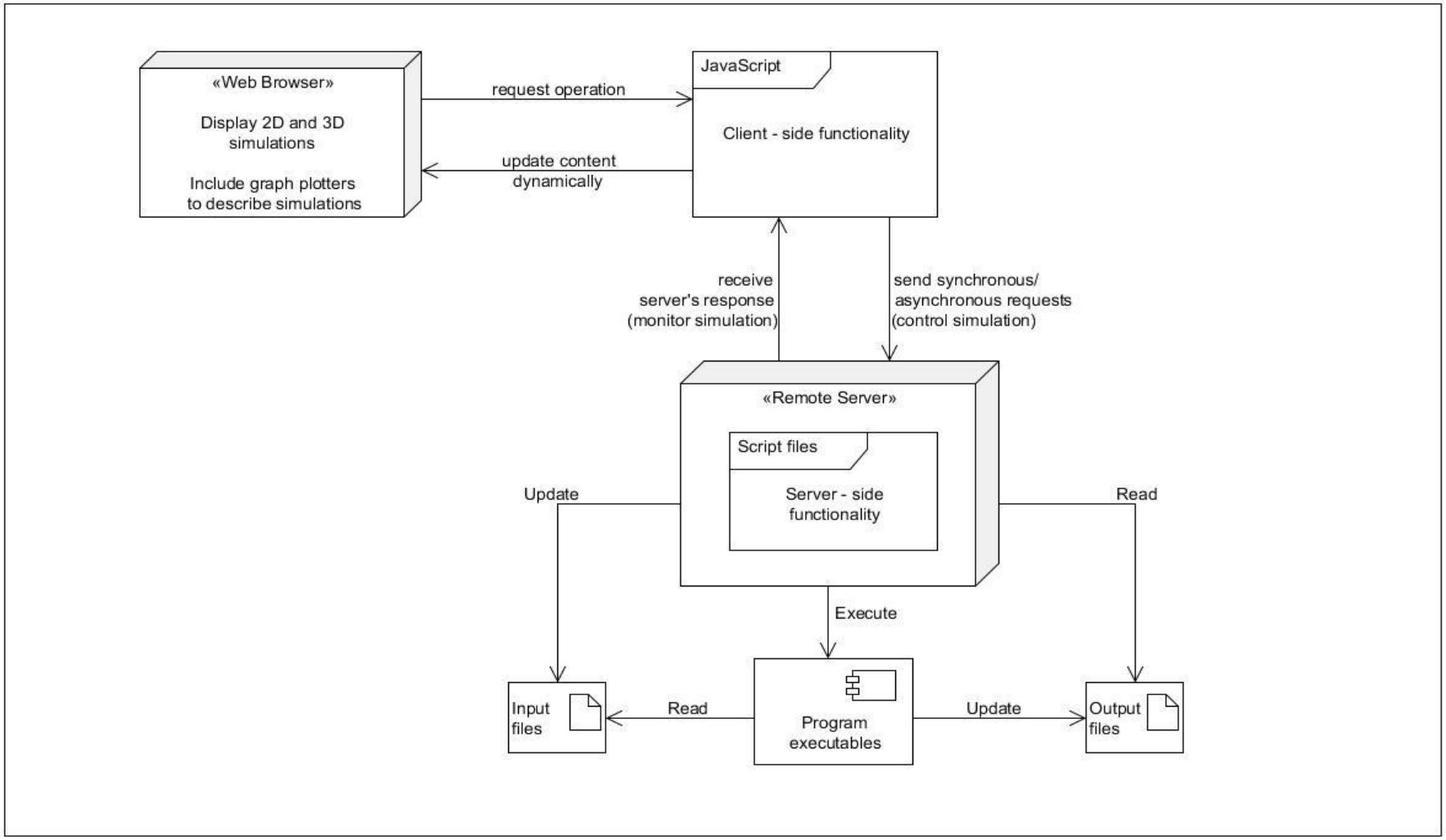



Figure 5.1: Model of the problem

## 5.2. System Architecture

The main aim of this project was to create a Web application where different dynamical systems would be presented inside the browser allowing users to experience their behaviour under various conditions. Apart from the dynamical systems assigned with this project, the system architecture should allow new systems to be added efficiently in the current model. Thus the Web application has been designed using the 3-tier architecture enabling the separation of its three different logics (presentation, business, and data). The next diagram describes the components of each tier and the relationships between them.

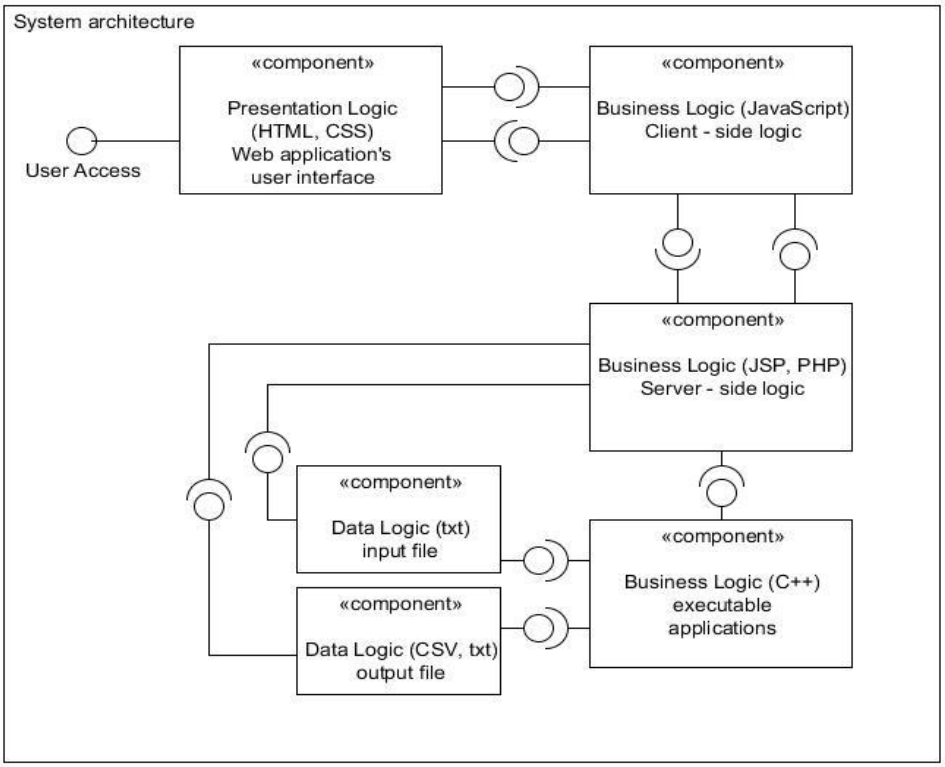



Figure 5.2: Component diagram

Presentation logic is the web interface of the application. It includes properties that enable the display of 2D and 3D simulations and a graph plotter to describe the simulation's behaviour. Along with the system's representation, each application provides various menus where users can modify the initial values of the system or change the plotter's options.

Business logic itself is divided into two subsections; client-side and server-side. Client-side handles any user requests made through the user interface and invoke server side operations where necessary. Server-side is responsible to handle the client-side's

requests and invoke the executable programs used to simulate the activity of the different dynamical systems. Since the executable programs are always located on the serve-side, they are included in the server-side business logic.

Data Logic includes all the input and output files required during a simulation program's execution. The information of the data logic can be accessed and manipulated from the server-side business logic including both script files and simulation programs.

Using the selected architecture, the system can be modified directly in each tier without affecting the rest of the system. In addition new dynamical systems can be added efficiently by including the new simulation programs on the server-side and adjusting the client side to display the simulation of the new system.

## 5.3. Technology Architecture

As mentioned earlier, the outcome of the project is a web application representing the run of different dynamical systems based on the research results of the CICADA developers. Users are able to interact with the visual representations of the systems via the available tools and control their activity. The new state of the system appears dynamically inside the browser. This chapter describes the technology used in the client and server sides to meet these project requirements.

The diagram below illustrates the technology implemented in each side.

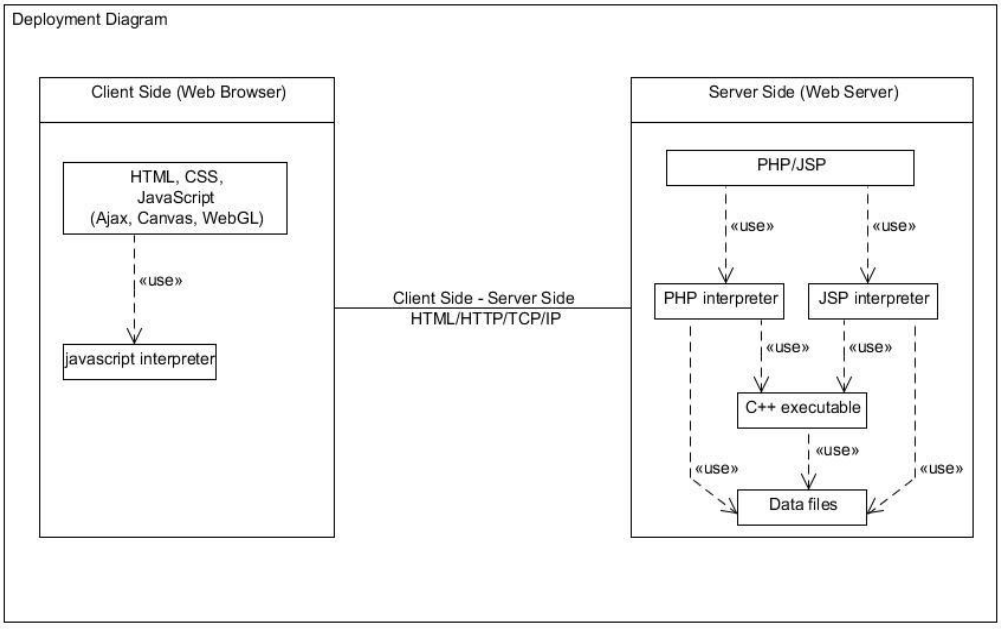



Figure 5.3: Deployment diagram

### 5.3.1. Client Side

The client side interface has been implemented using HTML and CSS. Thus the interface of the application is accessible through any internet browser running on a desktop or mobile machine. The applied technologies are supported by the latest versions of all frequently used internet browsers with the exception of WebGL which is not currently supported by the Internet Explorer.

A major task of the project was to create dynamic updated web pages and thus all of the Ajax technologies described in chapter 3.2 have been applied. JavaScript, the application logic behind Ajax, has been used to cover all the requirements of the client side functionality. This functionality includes the implementation of request-response messages using XMLHttpRequest required for the client-server communication and for the dynamic updating of the illustrated simulations.

The behaviour of the dynamical systems is illustrated using the Canvas element of HTML for 2D implementations and WebGL technology for the 3D implementations as described in chapter 3.4. Both Canvas and WebGL are implemented through JavaScript. The Graph plotters have been implemented using HTML elements and CSS styling properties while JavaScript handles the required functionality and update the graphs dynamically.

### 5.3.2. Server side

As explained before the behaviour of the dynamical systems included in this application will be controlled by different simulation programs that have been already implemented by CICADA developers. These programs will be located in the server-side. Thus, the role of the server side script will be to execute those programs using the parameters passed from the client's request. After the execution the output will be included in the server's response message.

Al the executable programs used in the implemented for the web application was built on C++ programming language. However this may not be the case for future simulation programs which CICADA may wish to add in the existing application. This will not be a problem since the server-side script is able to use any type of executable files.

For the implementation of the server-side functionality, two approaches have been tested during the design of the system. The first one was to use the PHP scripting

language to implement all the required functionality. The second approach was to implement the application on a TOMCAT server which supports java technologies on the server-side such as Servlets and JSP. It turned out that both of the approaches were able to call an executable file on the server. However the “exec” method of PHP which is responsible for program executions was not as flexible as the java alternative. More precisely “exec” was not able to control the program’s operations after the execution as it was just returning the produced output. JSP in the other hand is a complete java environment and thus is more capable to interact with the program while it is running.

Due to the nature of the executable programs used in this application, both PHP and JSP were able to cover the system’s requirements. This is because all the values required during the programs’ run are either passed as command line parameters when the program is called or through an input file. However the fact that the “exec” command of PHP has been disabled in some web-servers, initiate the need to focus on the JSP implementation of the server – side functionality.

In addition the web application should be able to adopt any further dynamical systems designed by CICADA. Since new executable files for those systems may require input values to be entered during the execution of the program, JSP is a more suitable approach to handle future upgrades on the system. PHP implementations can also be upgraded to handle new dynamical systems as soon as we modify the executable files so that they read input values as command line parameters or using input files.

Both PHP and JSP approaches have been implemented for each application in the system. They do carry-out the same functionality and they are both included in the source code of the web application. The reason that both of them have been included in the web application is to let the user decide on which type of server, PHP or JSP, wants to install the system.

The sequence diagram on the next page (figure 5.4) describes a sample run of the application after a user action. Client side sends a request to the server and updates the web page information dynamically after the server’s response. Server-side script in this example is implemented using JSP.

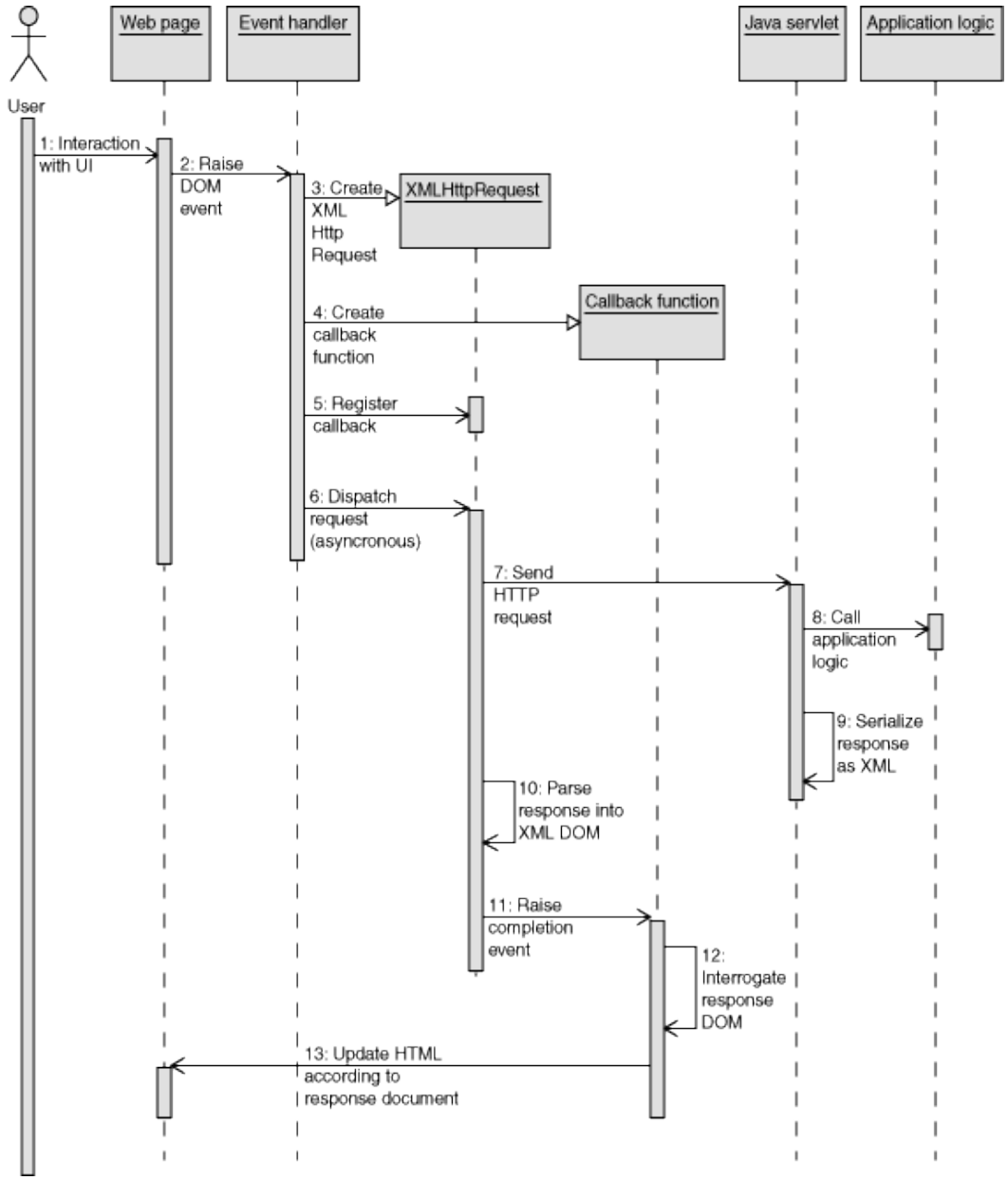



Figure 5.4: Sequence diagram

# 6. Implementation

This chapter explains how the technology described in chapter 3 has been combined together using the selected system architecture to implement the functionality of the web application.

## 6.1. Core Functionality

Both client and server side include some core functions which are responsible for the main operations of each side.

### 6.1.1. Client – side

Core function for the client side which is used in every application is the ServerRequest(). ServerRequest() has been implemented to cover the communication between the client and server sides. It includes all the operations related to the XMLHttpRequest of Ajax and in addition, it provides methods to select what parameter values to be send on the server-side for each request.

### 6.1.2. Server – side

Core functionality of the server-side is the three executable applications which are used to calculate the values used in each simulation. The three executables are dimred.exe for the dimension reduction application, singlePendulum.exe for the single pendulum application, and doublePendulum.exe for the 2D and 3D double pendulum applications. The dimred.exe and doublePendulum.exe produced from members of CICADA group, while doublePendulum.exe implemented from scratch for this project. On their initial form, executables given by CICADA were always returning the same output based on some hardcoded input values. Thus, some modifications had to be made to allow the executables to receive several input values. Taking in count that the application must be able to accept any kind of executable applications, the three executables have been modified to read input values and return output values in three different ways. The dimred.exe application has been modifies to read values from a text file and output the result on a different text file. singlePendulum.exe reads input values as command line parameters and outputs the result as normal output (i.e. using the “cout<<” command). The doublePendulum.exe reads input values as command line parameters and outputs the result in a CSV file (initial output method). The server-side script has been implemented to supply all three cases.

## 6.2. User Interface

The figure below illustrates the main page of the web application inside the browser.

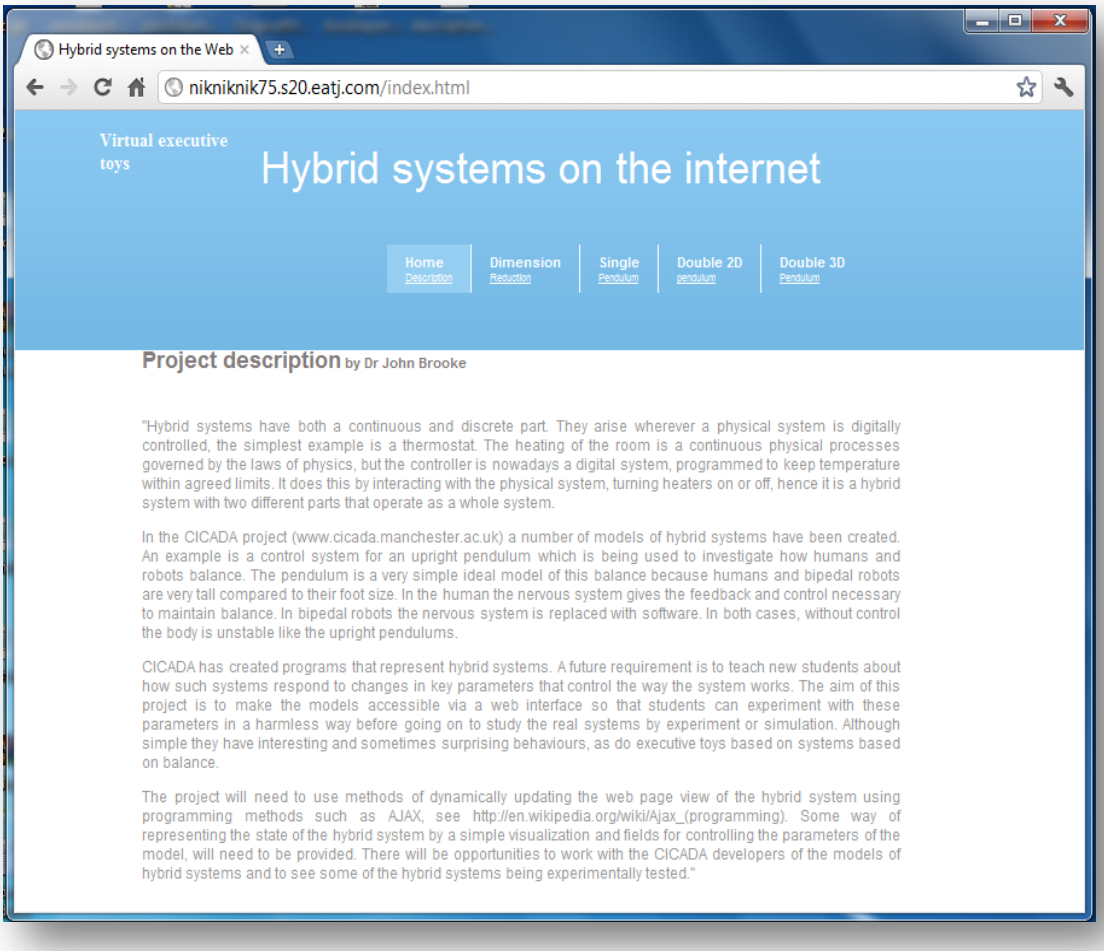
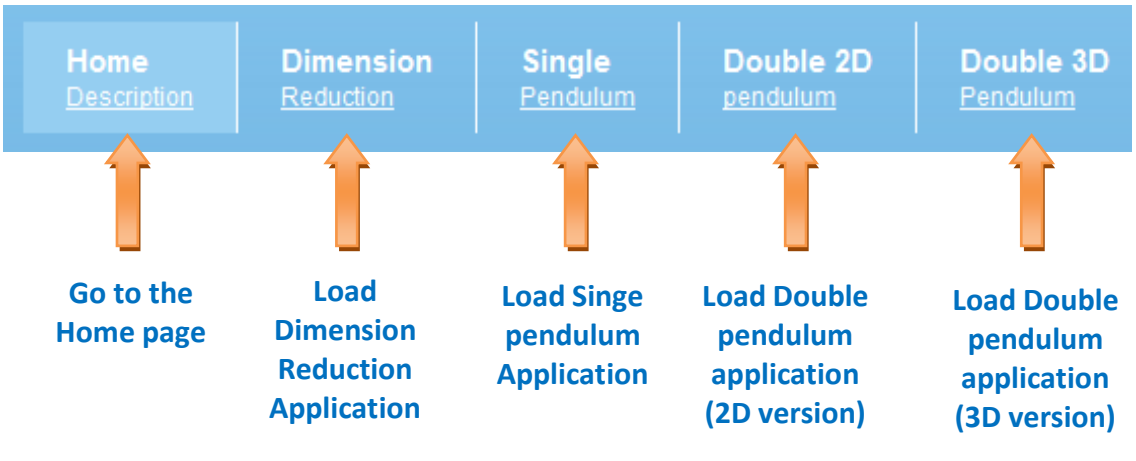

Hybrid systems on the Web
nikniknik75.s20.eatj.com/index.html
Virtual executive toys
Hybrid systems on the internet
Home
Description
Dimension
Reduction
Single
Pendulum
Double 2D
pendulum
Double 3D
Pendulum
Project description by Dr John Brooke
"Hybrid systems have both a continuous and discrete part. They arise wherever a physical system is digitally controlled, the simplest example is a thermostat. The heating of the room is a continuous physical processes governed by the laws of physics, but the controller is nowadays a digital system, programmed to keep temperature within agreed limits. It does this by interacting with the physical system, turning heaters on or off, hence it is a hybrid system with two different parts that operate as a whole system.
In the CICADA project (www.cicada.manchester.ac.uk) a number of models of hybrid systems have been created. An example is a control system for an upright pendulum which is being used to investigate how humans and robots balance. The pendulum is a very simple ideal model of this balance because humans and bipedal robots are very tall compared to their foot size. In the human the nervous system gives the feedback and control necessary to maintain balance. In bipedal robots the nervous system is replaced with software. In both cases, without control the body is unstable like the upright pendulums.
CICADA has created programs that represent hybrid systems. A future requirement is to teach new students about how such systems respond to changes in key parameters that control the way the system works. The aim of this project is to make the models accessible via a web interface so that students can experiment with these parameters in a harmless way before going on to study the real systems by experiment or simulation. Although simple they have interesting and sometimes surprising behaviours, as do executive toys based on systems based on balance.
The project will need to use methods of dynamically updating the web page view of the hybrid system using programming methods such as AJAX, see http://en.wikipedia.org/wiki/Ajax_(programming). Some way of representing the state of the hybrid system by a simple visualization and fields for controlling the parameters of the model, will need to be provided. There will be opportunities to work with the CICADA developers of the models of hybrid systems and to see some of the hybrid systems being experimentally tested."

Figure 6.1: Main page of the application

### Main menu

From the main page of the system, users can load any of the four applications available. In the top side of the page someone can recognise the menu where users of the system can select which application they would like to use. The menu options are shown below.



### 6.2.1. Dimension Reduction application

Once a user selects to load the dimension reduction application, the initial page of the application will be displayed in the browser as illustrated in the figure below.

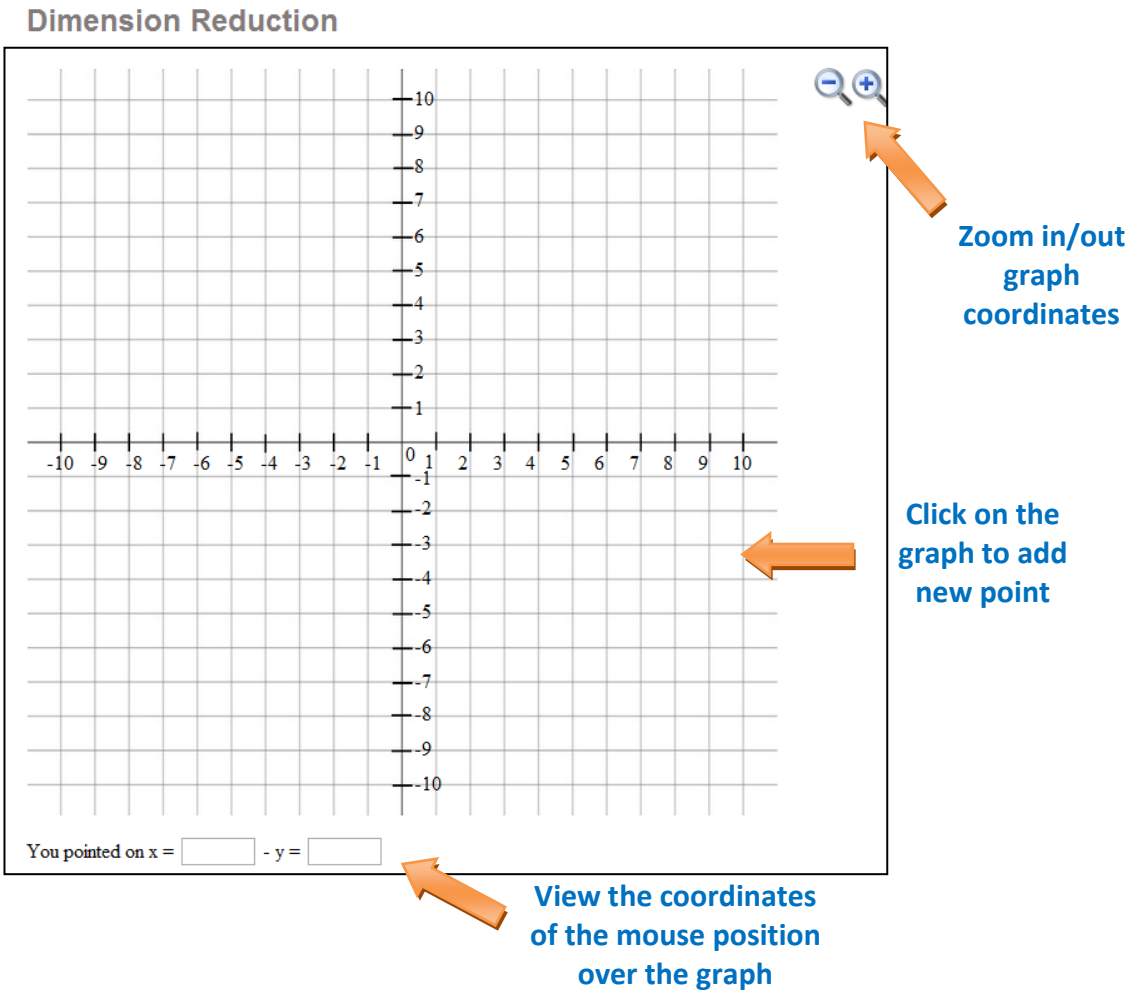

Dimension Reduction
Zoom in/out graph coordinates
Click on the graph to add new point
You pointed on x =
- y =
View the coordinates of the mouse position over the graph

Figure 6.2: Dimension reduction application – initial page

Before the user starts adding points in the graph, he/she should first set the coordinates of the graph so that he/she can add all the required points. This is done using the two zoom buttons located on the top right corner of the application. Each time the user presses the zoom out or zoom in button, the coordinates of the graph are multiplied by 2 or divided by 2 respectively. The initial coordinates of the graph are from -10 to 10 for both of the axes as shown in the figure above. So if the user clicks on the zoom in button once, the new coordinates will be -5 to 5. Similarly, using the zoom out button, the coordinates will be from -20 to 20. The method responsible to change the axes' values in called fixnums(). It is being executed after every zoom operation and it manages to adjust the graph coordinates using a variable to hold the current zoom factor of the graph.

Once the user has set the required coordinates, he/she can start adding points on the graph and see their representation in one dimensional space. To help the user to insert the required points for dimension reduction, the application provides two additional boxes displaying the exact X and Y positions of the mouse pointer over the graph. Two functions named pointToX(px) and pointToY(px) are used to translate the left and top pixel positions of the mouse pointer respectively. Each function takes the pixel position as a parameter and translates it into graph position multiplied by the zoom factor value. Note that the pixels are counted having as initial point (0, 0) the top left corner of the graph, while the graph uses the origin of the axes as its initial point. PointToX() and PointToY() functions are available in appendix A.2.

After some point additions, the application will look like in the figure below.

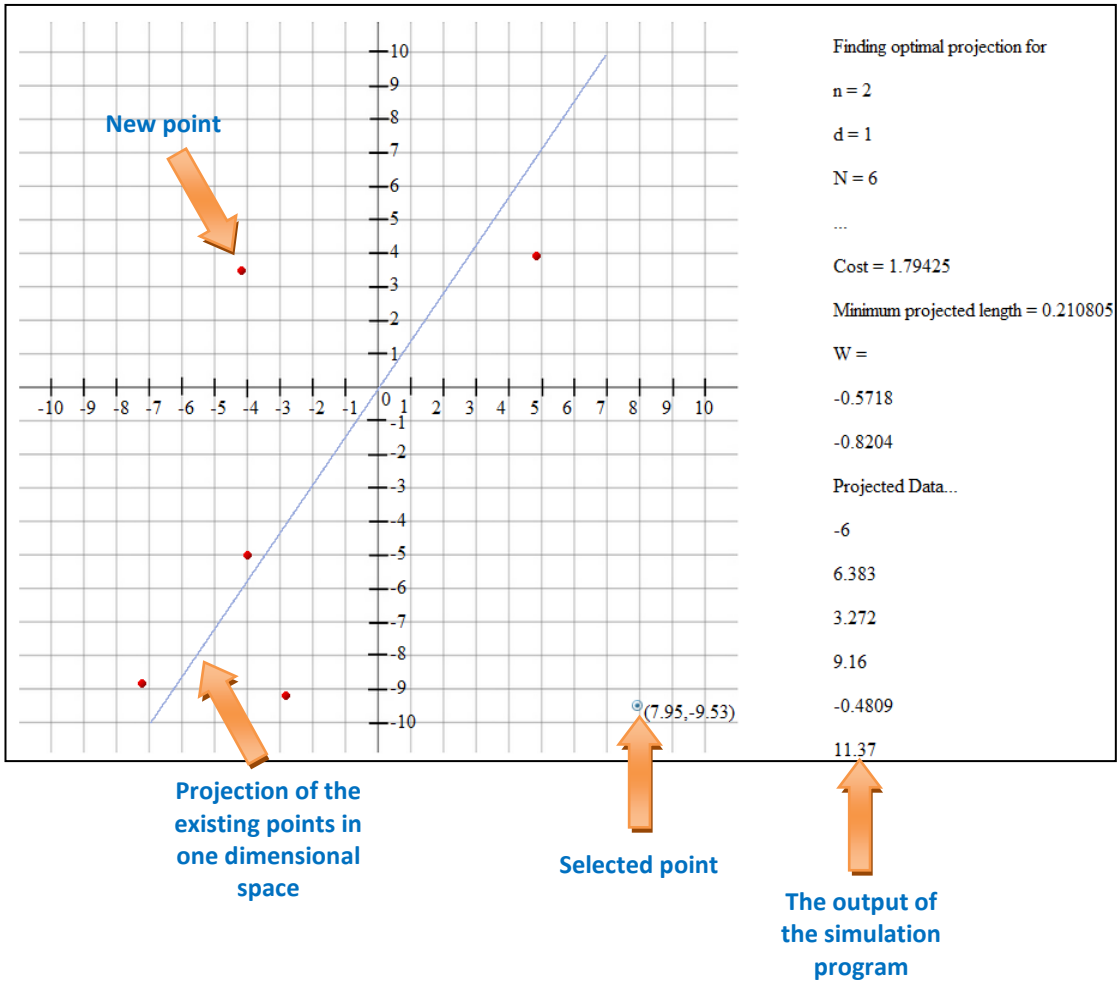



Figure 6.3: sample run of the dimension reduction application

Each time the user clicks on the graph implementation, the application uses the ServerRequest object to send a new XMLHttpRequest to the server. As parameters in the request it includes all the existing points of the graph. The server script with its

turn updates the relevant input file and executes the simulation program. The result of the execution is stored in the output file. After the execution, the server script manages to read the content of the output file and send it back to the client-side. The client side then dynamically updates the content of the page to display the projection of the current points in one dimensional space. In addition, the full content of the output file, as returned from the server appears on a panel next to the graph displaying all the details of the execution.

The new added point is represented using a red point icon. This icon is an HTML image element which is placed inside the graph dynamically each time the user clicks on a new position. The X and Y positions of the point are assigned as attributes of the new element along with a point id value to reference the element from the DOM. When the user moves the mouse pointer over an existing point, the exact position of the point appears on its left side while the source of the image is temporary changed to display the selected point in using a blue point icon as shown in the previous example.

In order to understand how the projection line is generated inside the graph we must first have a look on the output values returned after the execution of the server-side program. Using the same example as above the first lines of the output file will be:

Finding optimal projection for

n = 2 ⟵ **The given points are defined in two dimensional space (2)**

d = 1 ⟵ **The output points will be defined in one dimensional space (1)**

N = 6 ⟵ **Number of points added for dimension reduction (6)**

The next part of the output contains the calculation results of the program as shown below.

Cost = 1.79425

Minimum projected length = 0.210805

W =

-0.5718

-0.8204

The cost and the minimum projected length are numbers which quantify how 'good' the projection is for the given data (closer to 1 the better). The two entries of the matrix W can be interpreted as the x and y components of a vector which points in the direction of the straight line onto which the data is being projected. To project the data points onto this line, each 2D point is multiplied by the transpose of W. For example if X is a two dimensional data point then X' = W.transpose() * X, where X' will be a one dimensional data point (a position on the line).

This calculation is performed for every given two dimensional point. The result of these calculations is included in the last part of the program's output as shown below.

```
Projected Data...
-6
6.383
3.272
9.16
-0.4809
11.37
```

The line representing the given points in one dimensional space should pass through the origin of the axes. The equation of the line is of type y=a (x) where "a" is given from the two elements of the matrix W such as (a=W1/W0). W0 and W1 are the first and second elements of the matrix W respectively. If the W0 has a value of zero, then the line is drawn on top of the Y axis.

The two end positions of the visible part of the line, inside the graph, are calculated using the equations y1 = a*(x_max) and y2 = a*(- x_max) where x_max is the maximum value displayed on the X axis. The coordinates of the two end positions will then be (x_max, y1) and (-x_max, y2). When y1 and y2 are greater than y_max (maximum value on Y axis) then the end positions are not on the top and bottom edges of the graph but on the left and right ones instead. In this case the end positions are calculated using the equations x1 = (y_max)/a and x2 = (-y_max)/a. The coordinates of the end positions will then be (x1, y_max) and (x2, -y_max). The line is then displayed inside the graph from one point to the other. The function addPoint() which is responsible to draw the projection line after each new point addition is available in appendix A.3.

### 6.2.2. Single pendulum application

When the user selects to launch the single pendulum application the main page of the application appears in the browser as shown in the figure below. The main page is divided in three parts; the canvas panel on the left, a graph plotter on the right and an options panel on the bottom.

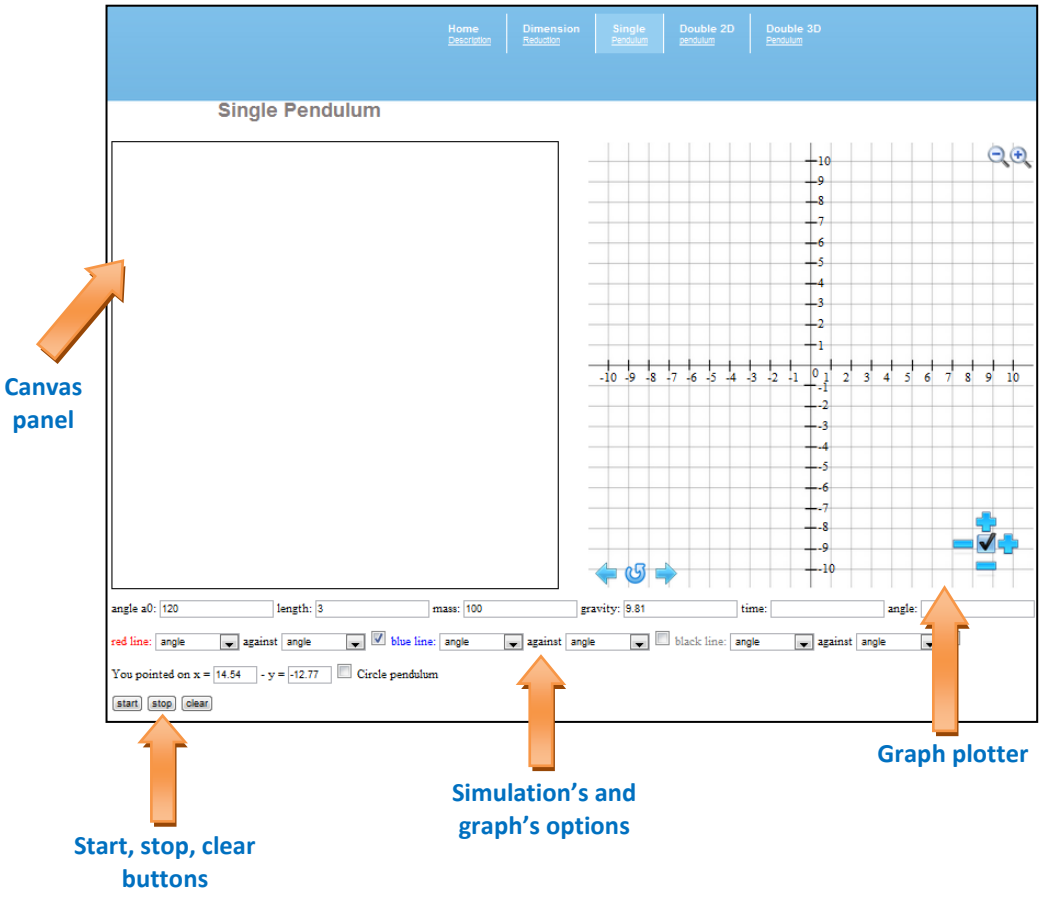
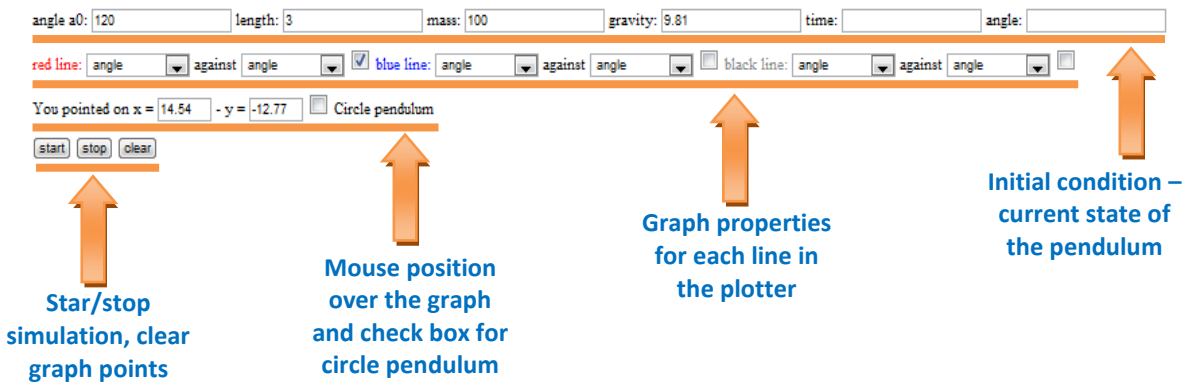



Figure 6.4: Single pendulum application – initial page

#### 6.2.2.1. The options panel

Before starting the simulation, the user should set the initial condition of the pendulum as well as the graph values to be drawn using the panel located at the bottom part of the page. Each line of the panel is described below.


angle a0: 120
length: 3
mass: 100
gravity: 9.81
time:
angle:
red line: angle against angle
blue line: angle against angle
black line: angle against angle
You pointed on x = 14.54 - y = -12.77
Circle pendulum
start stop clear
Star/stop simulation, clear graph points
Mouse position over the graph and check box for circle pendulum
Graph properties for each line in the plotter
Initial condition – current state of the pendulum

The first line contains four input boxes where users can set the initial condition of the pendulum. This includes the initial angle (a0) of the pendulum, its mass, the length of the rod and the gravity values. Next to these input boxes there are two additional boxes to display the time and angle values during the run of the pendulum.

The second line is used to set the graph values which will be described in the plotter. The plotter supports up to three different graphs to be drawn during the run of the pendulum using red, blue and black colours for the first, second and third graphs respectively. User is able to choose the values to be described in the X and Y axis of each graph as well as select which graphs will be displayed in the plotter using the relevant check box for each graph.

The third line includes two boxes displaying the X and Y coordinates of the mouse pointer over the graph in the same way as implemented in the dimension reduction application. Next to them there is a check box used to indicate whether the pendulum follows a continuous circle motion if the initial angle is set to 180° or it keeps swinging.

The last line provides three buttons. The first two are used to start and stop the pendulum's simulation while the third one clears the content of the graph plotter. The reason that the "clear" button has been placed near the start and stop buttons is that users may need to remove any existing information from the graph before starting a new simulation.

#### 6.2.2.2. Draw simulation

When the user clicks on the start button for the first time the system invokes a function called pendulum_start(). This function uses the ServerRequest method to forward an XMLHttpRequest to the server including the parameters required for the pendulum's simulation (i.e. a0, mass, length, gravity). The values are then passed on the server side executable which calculates the simulation values for a period T of the pendulum. The output is returned back to the client-side. Since there is no friction force applied on the implemented simulation of the single pendulum, the behaviour of the pendulum will be the same in each period. The output values received from the server are stored in the form of an array were each line describes a different state of

the pendulum. The last line of the pendulum_start() function calls the draw() method which is responsible to draw the pendulum inside the canvas element.

Canvas and WebGL technologies do not include any primitive methods to construct animating shapes. In order to construct an actual animation, such as the moving pendulum for our case, the canvas panel needs to be cleared and the previous shapes to be redrawn in new positions. The draw() method is responsible to clear the canvas element and redraw the pendulum in its next state. To display an animating scene of the simulation the draw() method is set to be executed after every fixed time intervals. This is achieved using the setInterval command available in JavaScript. So instead of just calling the draw() in the last line of the pendulum_start() function, the draw() method is set to be executed every 50ms using the following command.

```
1.      u = setInterval("draw()",50)
```

After the start of the simulation the application will look like in the figure below.

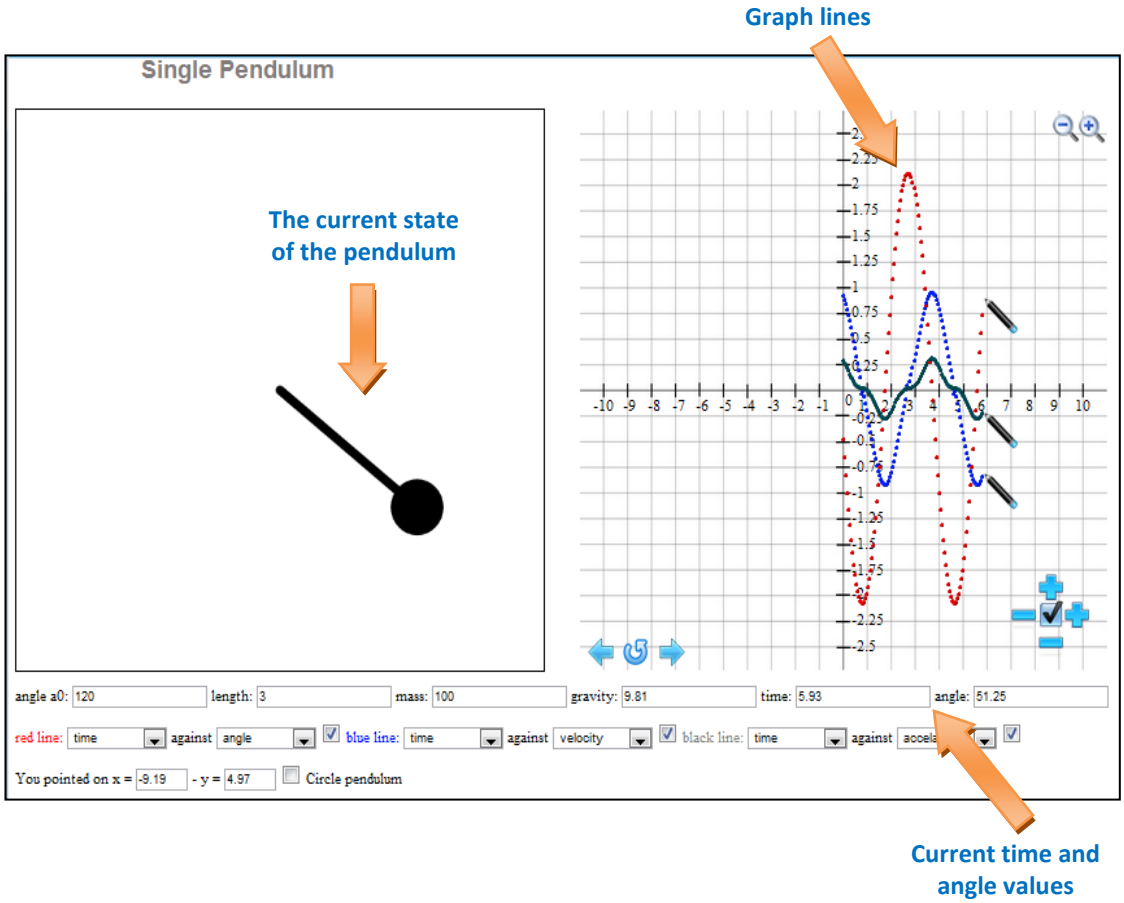



Figure 6.5: Single pendulum application – simulation in operation

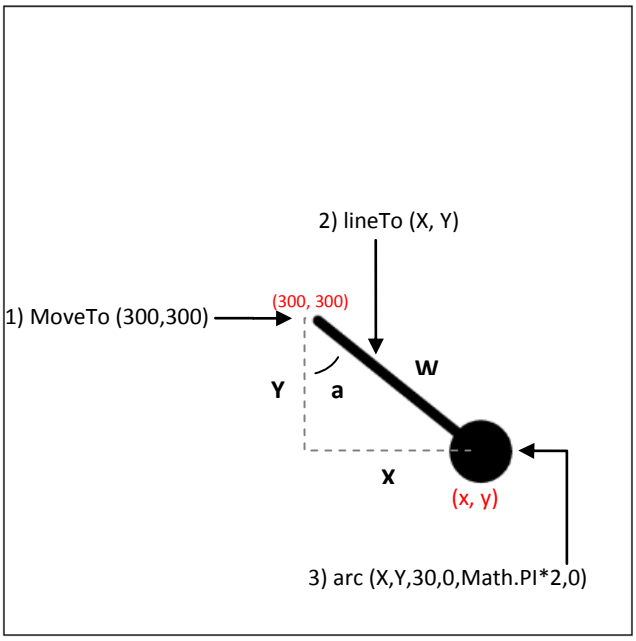



Figure 6.6: Single pendulum – draw the scene

The figure shown above illustrates how the draw() method manages to draw the pendulum inside the canvas panel. As mentioned earlier, the output of the simulation values is stored in an array. Each time the draw() method is being executed, a pointer variable is increased by one to point on the next position of the array (i.e the next state of the pendulum). One column of the array holds the angle of the pendulum an each state. Using this angle and the length of the pendulum's representation (w) which for display purposes is always the same inside the canvas (200px), the X and Y coordinates of the pendulum are being calculated using the equations:

x = w * Math.sin(a)
y = w * Math.cos(a)

Having these two values, the pendulum is drawn using the three methods of the canvas element shown in the figure above. Using the moveTo(300,30) command, the drawing position moves at the centre of the canvas element since the canvas size is set to 600 X 600. Then the LineTo() and arc commands draw the rod and sphere shapes respectively using the (X,Y) coordinates. The code for the draw() method is available in appendix A.4.

The stop button actually pauses the simulation using the clearInterval() command to stop the draw() method of being executed. Using the start button the user can later resume the activity of the simulation. User can change the initial condition of the

simulation during the run of the pendulum or while it is at pause position. The pendulum_start() function will then be executed again as described above sending the new values on the server-side to receive the new output values based on the current initial condition. The new output will then be used to draw the simulation. The graph plotter appearance and functionality will be described in chapter 6.2.5.

### 6.2.3. Double pendulum application (2D)

The user interface of the double pendulum (2D) application is illustrated below.

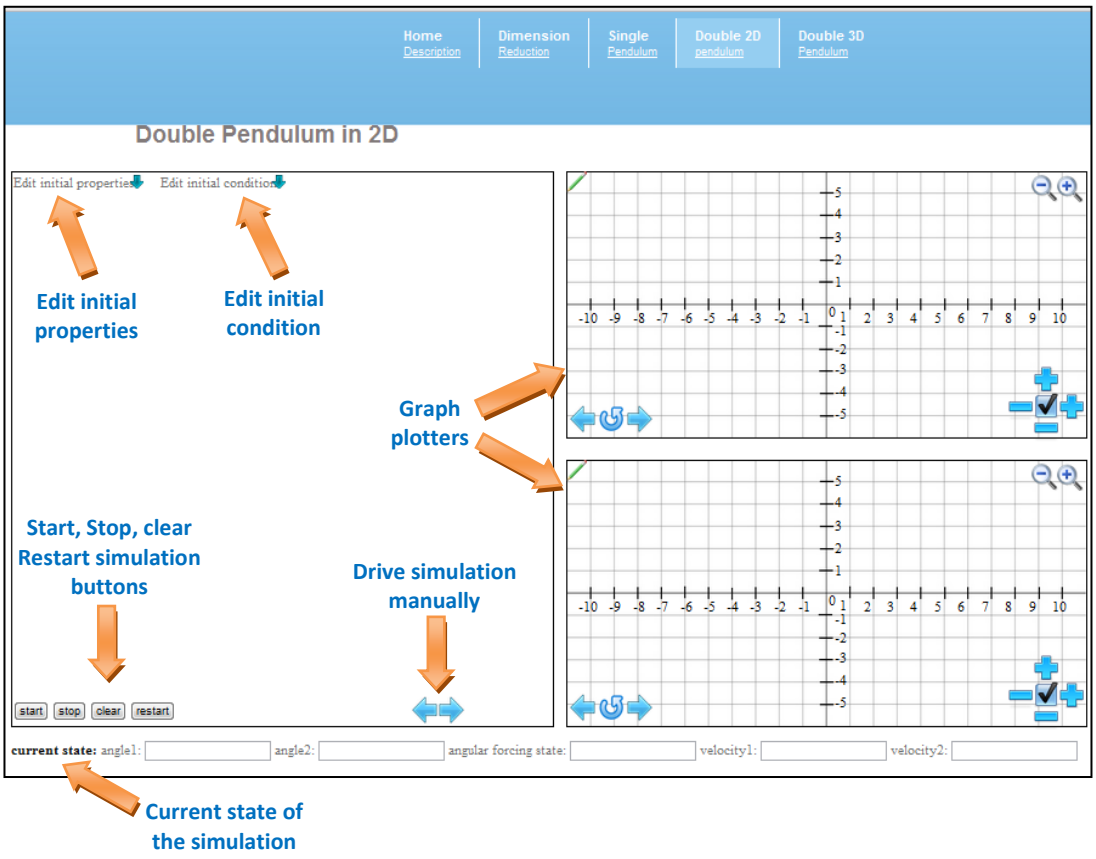



Figure 6.7: Double pendulum application 2D – initial page

#### 6.2.3.1. New features

The interface of the double pendulum application, curries the same functionality with the single pendulum implementation described earlier. In addition, this application provides some extra features as shown in the figure above.

A main feature for this application is the second graph plotter which has been added below the first one, on the right side of the canvas element. The two graphs are used to describe the behaviour of both the top and the bottom pendulum simultaneously.

Each plotter can display up to three lines illustrating the graph values selected by the user.

The new “restart” button located next to the “start”, “stop” and “clear” buttons is used to reset the simulation of the system to its initial state. To achieve this, the array pointer used for the array holding the output values returned from the server, is set to zero.

The two arrow-style buttons added inside the canvas element allow users to interact with the simulation while it is at pause position. More precisely using the left button, the application displays the pendulum on its previous state while using the right button it displays the next state of the pendulum. This is done by calling the draw() method once each time one of the two buttons is clicked. To move the simulation one state backward, the array pointer value is decreased by one before calling the draw() method. The graph values are also modified to describe the current state of the pendulum each time the view previous or next state operations are being executed.

The two menu buttons located on the top right corner of the canvas element are used to view and modify the initial properties and initial condition and of the simulation. The two menus are being displayed inside the canvas element as shown below.

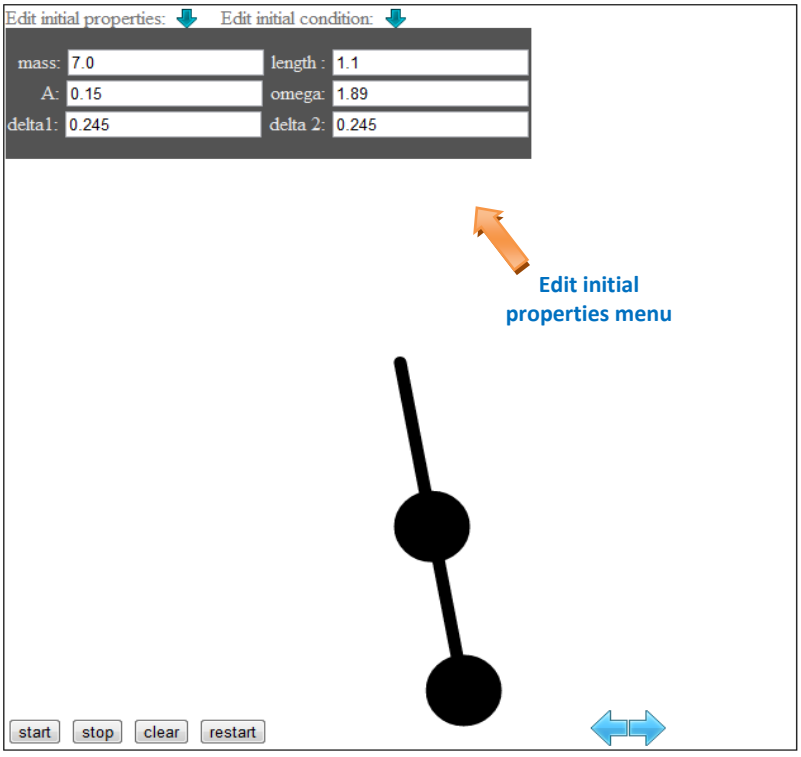



Figure 6.8: Double pendulum application 2D – initial properties menu

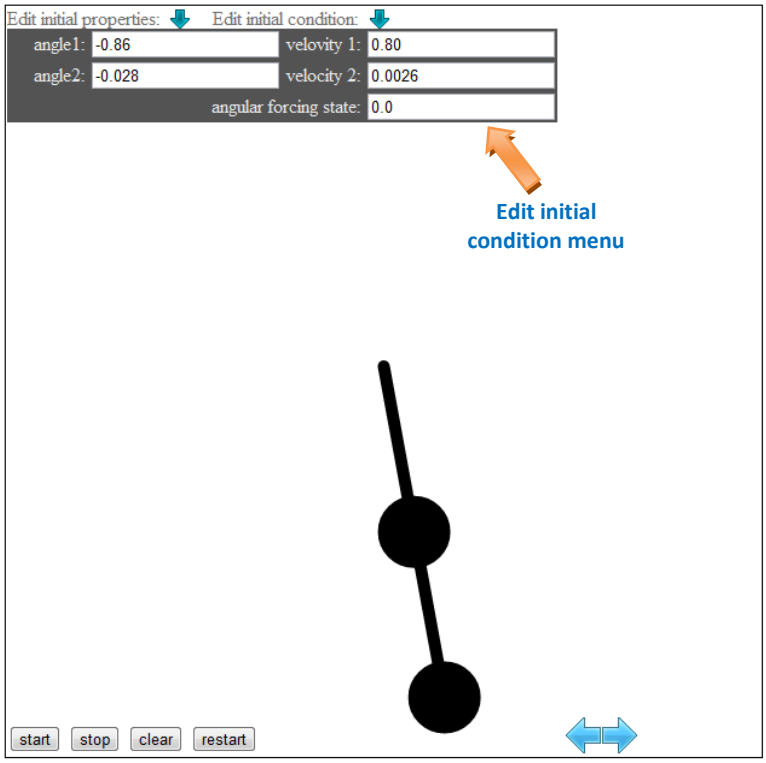



Figure 6.9: Double pendulum application 2D – initial condition menu

As initial condition of the simulation are considered the initial angle and initial velocity of each pendulum as well as the angular forcing state. The initial properties of the simulation include the mass, length, A, omega, delta1 and delta2 values. The mass and the length of the rod are always equal for the top and bottom pendulums so there is only one input box available for each attribute in the "edit initial properties" menu. Delta1 and delta2 values are used to specify the friction force applied on the upper and lower pendulum respectively. A and omega values are related to the angular forcing state of the pendulum.

On the lower part of the page there are several values used to display the current state of the pendulum during its run. These values include the current angle and velocity of each pendulum and the angular forcing state.

The execution of the server-side program which simulates the run of the double pendulum takes about ten seconds to calculate the result values for the simulation. This is a relatively large time period for the user to wait for the application's response after he/she clicks on the start button or after he/she changes a value from the initial properties or the initial condition menu. To overcome this, the application forwards an asynchronous request to the server using the XMLHttpRequest object. Asynchronous

request means that the execution of the client-side will not wait for the server's response to continue with the rest of its operations; as happens with a synchronous request. So even if the execution of the simulation program has not been completed, the application can start drawing the simulation using the first output values of the execution until the complete output is finally returned to the client-side.

### 6.2.3.2. Geometry of the double pendulum

As in the simple pendulum application, the output of the program's execution is stored in an array. The output values used to draw the simulation are the two angles, θ and ϕ of the top and bottom pendulum respectively. In the double pendulum simulation used for this application, the bottom pendulum is fixed to swing in a direction perpendicular to the motion of the top pendulum. For example, if the top pendulum is set to move in left - right directions, the bottom pendulum will always move backwards and forwards. This geometry of the double pendulum used in this application is described in the figure below [38].

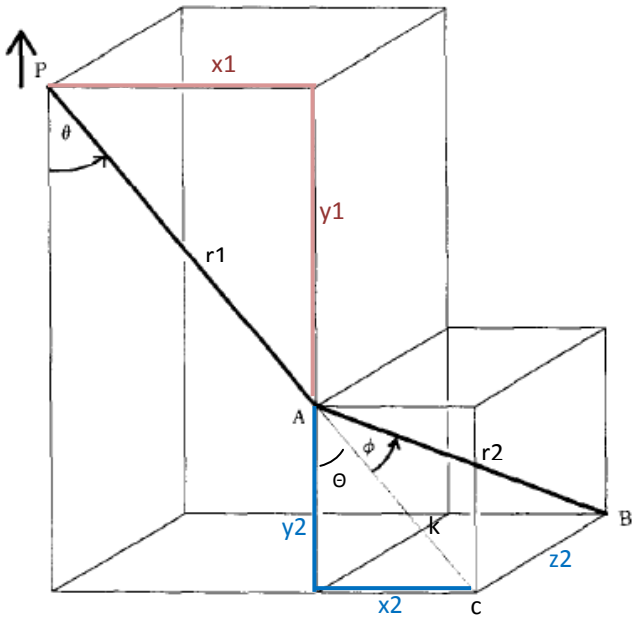



Figure 6.10: Geometry of the double pendulum [38]

In order to understand the above figure is important to remember that the lower pendulum is always perpendicular to the upper pendulum. The position of the top pendulum is calculated using the angle θ and the length w of the pendulum. The position of the bottom pendulum is calculated using both θ and ϕ angles. It can be thought as an extension of the top pendulum, based on the same angle θ, which is also

being rotated along the Z axis using the angle ϕ. Thus, the angle θ is used to describe the motion of the bottom pendulum in the left-right direction while the angle ϕ describes its backward and forward motion.

As mentioned earlier, the length of the two pendulums is the same (i.e. r1=r2). For the representation of the simulation inside the canvas panel, the two values r1 and r2 have been set to 150px. The pixel size is only used for the representation of the simulation and thus it is does not affect the calculation of the simulation values. Applying the relevant trigonometry equations, in reference to the figure 6.10, the positions of the two pendulums in each state are calculated as follows:

r1 = r2 } Length of the pendulum

x1 = r1 * Math.sin (θ)
y1 = r1 * Math.cos (θ)
z1=0 } Top pendulum's position values

k = r2 * Math.cos (ϕ) } Intermediate step

z2 = r2 * Math.sin(ϕ)
x2 = k * Math.sin (θ)
y2 = k * Math.cos (θ) } Bottom pendulum's position values

The positions of the two pendulums will be:

Top pendulum: (x1, y1, z1)
Bottom pendulum: (x2+x1, y2+y1, z2)

### 6.2.3.3. Drawing the simulation

The simulation of the 2D double pendulum is implemented using the canvas element. Because of the nature of this double pendulum simulation, it is not easy to observe the activity of both pendulums in a two dimensional space. To overcome this, the application provides the option to modify the viewing angle of the simulation using an option box which is visible after the start of the application.

After the user clicks on the start button, the simulation of the double pendulum will be displayed inside the canvas element and the graph plotters will start drawing the relevant lines as displayed below.

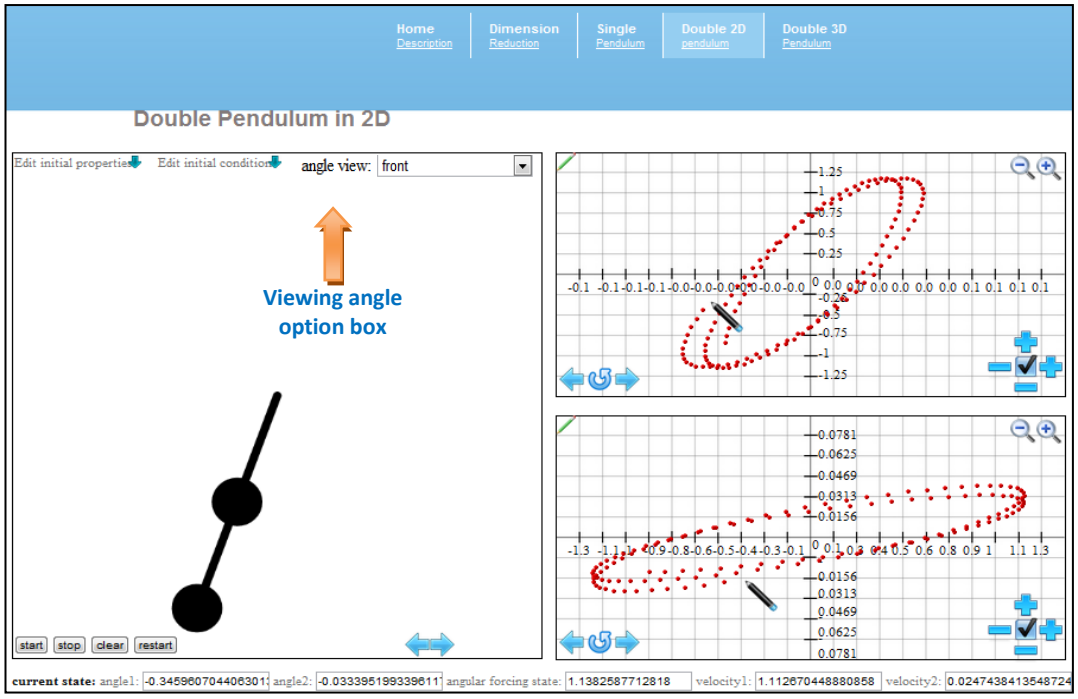



Figure 6.11: Double pendulum application (2D) – simulation in operation

The viewing angle option box contains 3 options; view the pendulum from the front position, view the pendulum from the right position or view a virtual display of the simulation where the motion of the bottom pendulum is in the same direction with the top pendulum (i.e. not perpendicular to the top one). The following examples describe how the double pendulum simulation has been implemented in each case.

**Front view of the simulation:**

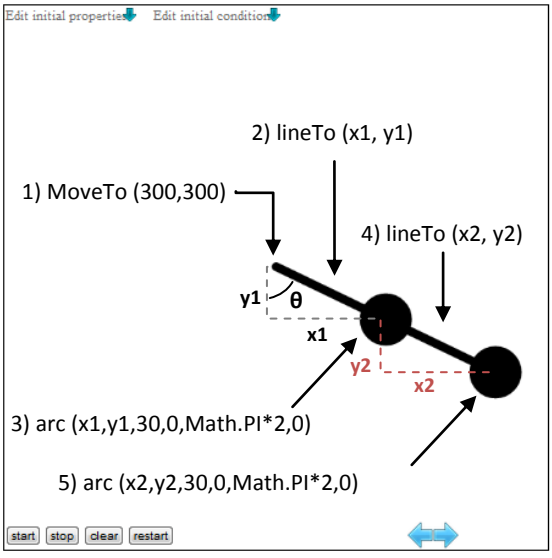



Figure 6.12: Front view of the double pendulum 2D application

**Right view of the simulation:**

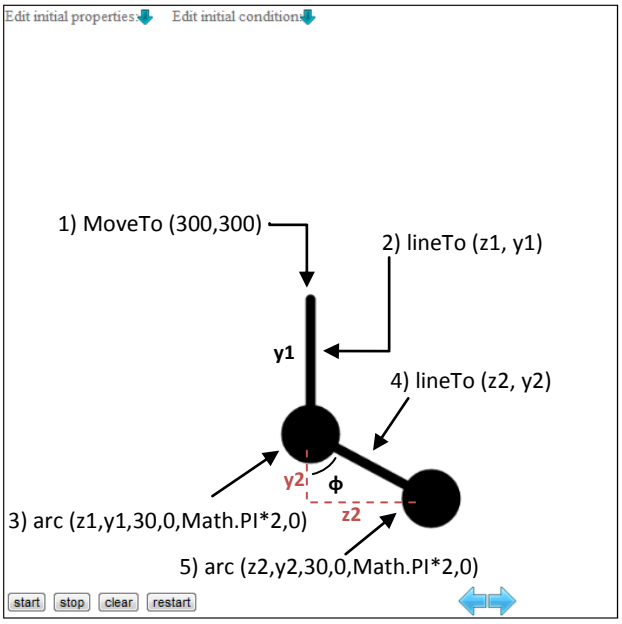



Figure 6.13: Front view of the double pendulum 2D application

**Virtual view of the simulation (not perpendicular):**

The x3 and y3 values used in this simulation are calculated as following:

x3 = r2 * Math.sin (ϕ)

y3 = r2 * Math.cos (ϕ), where r2 is the length of the bottom pendulum

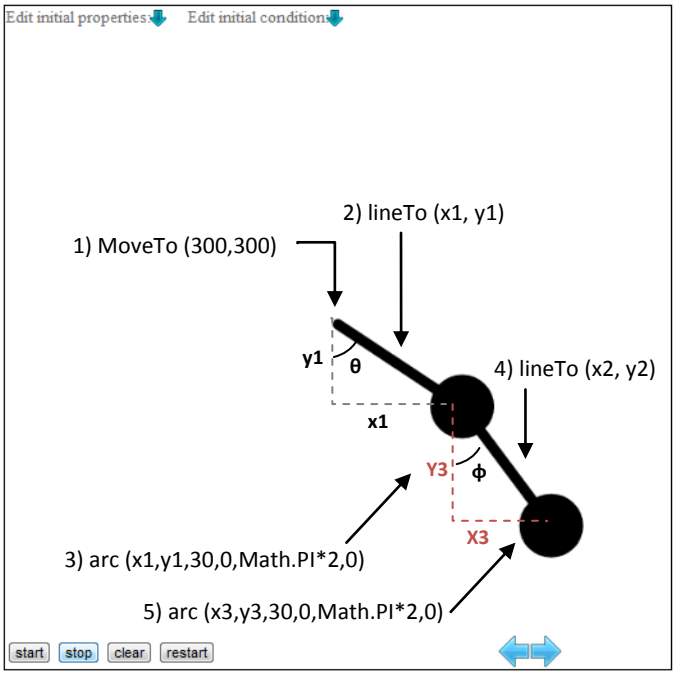



Figure 6.14: Front view of the double pendulum 2D application

### 6.2.4. Double pendulum application (3D)

The user interface of the 3D version of the application as well as the provided functionality cover all the features described in the previous 2D application. The different part is that the simulation is generated using the WebGL technology instead of canvas. As mentioned earlier on the report, WebGL allows 3D graphics to be displayed inside the canvas element. The only difference on the user interface of this application is that it includes five new buttons used to rotate the scene in alternative viewing angles as well as to enable or disable autorotation of the pendulum along the Y axis. The initial page of the application with the new buttons is displayed in the figure below.

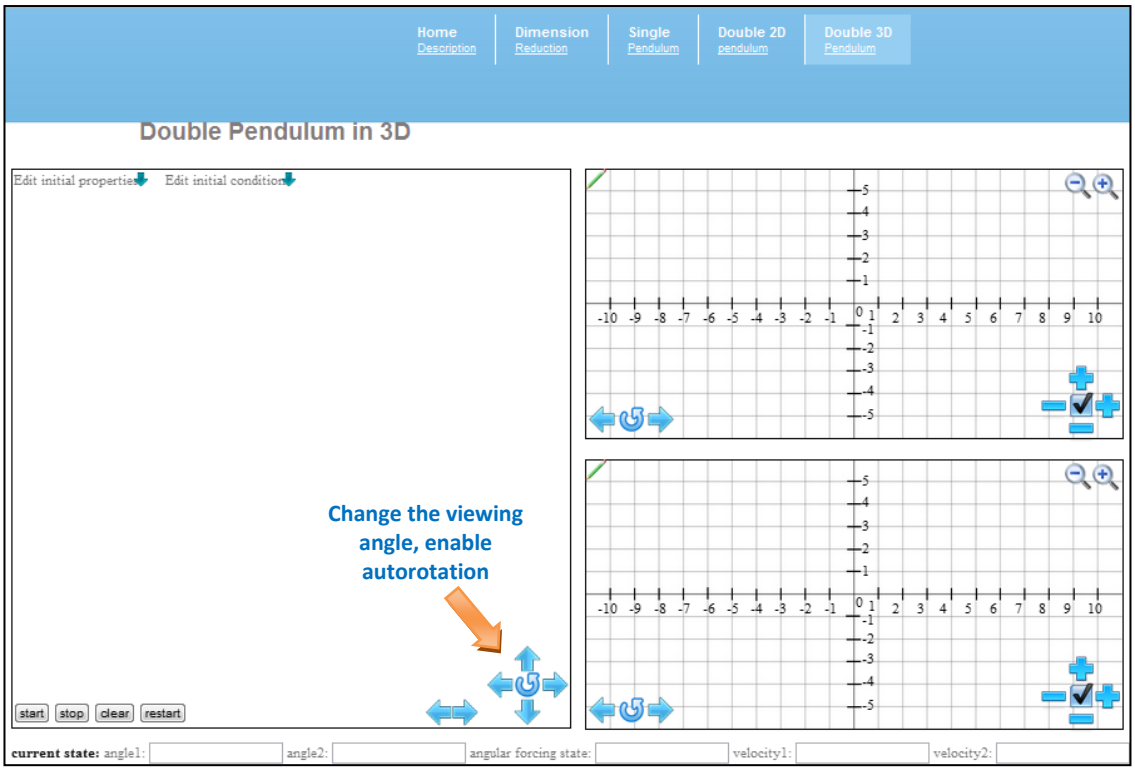



Figure 6.15: Double pendulum application (3D) – initial page

As described in chapter 3.4.2, the information used to define the shapes of a 3D scene in WebGL, is stored in the form of an array matrix. Matrices are ideal for this purpose since any transformation on the shapes can be described by multiplying the array matrix with a new matrix which holds the transformation details. Transformations include changes on the position of the shape inside the 3D scene or rotation of the shape around the X, Y or Z axis.

OpenGL (not WebGL) provides some commands which are used to apply transformations on the buffer matrix before adding each shape on the scene.

Unfortunately, the same commands are not provided in WebGL. To overcome this, there is a JavaScript library called "glMatrix" available in Google's repository that enables the same functionality to be used in WebGL applications. A sample use of the translate and rotate commands used to move and rotate the shape respectively is shown below.

```
1.      matrix.translate("inputmatrix", [X,Y,Z];
2.      matrix.rotate("inputmatrix", angle, [1,0,0]);
```

The translate command takes as parameters the matrix to apply the transformations on and the new position of the shape. The rotate command takes as parameters the matrix, the angle of the rotation and the axes where the rotation will be performed around, denoted by the value 1.

As previously mentioned, the primitive shape of WebGL is the triangle. For this reason, the pendulum has been implemented in the form of a diamond consisting of eight subsequent triangle shapes. Four of them are used in the upper part of the diamond and four in its lower part. The rod part of the pendulum has been implemented as a line with length 150 pixels.

When the user clicks on the start button the 3D scene representing the double pendulum's simulation is being displayed as shown in the following figure.

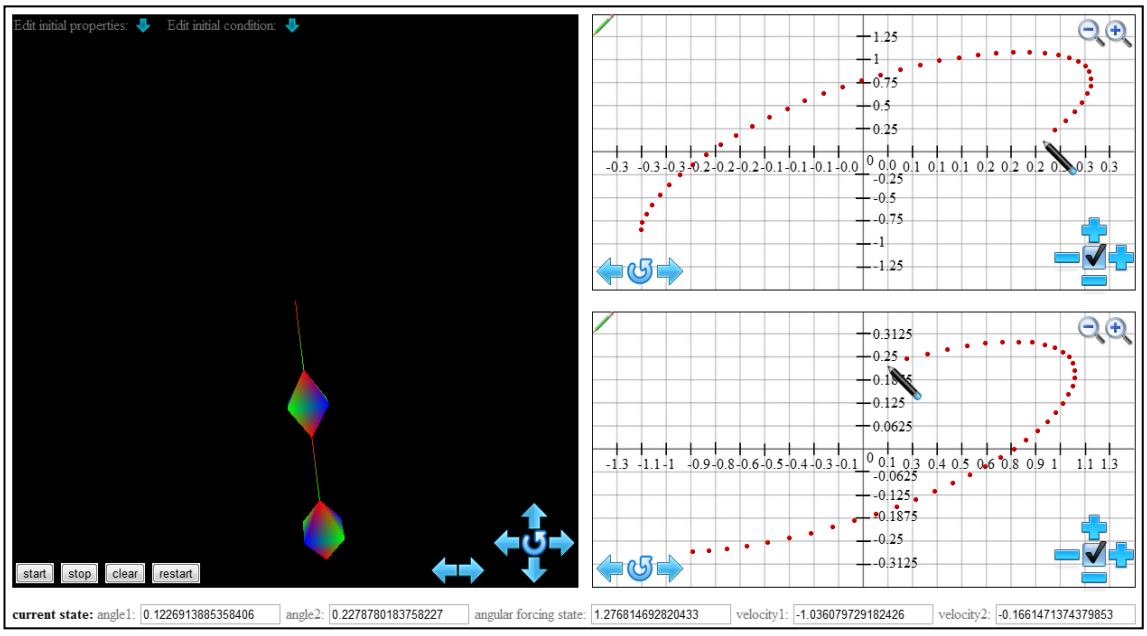



Figure 6.16: Double pendulum application (3D) – simulation in operation

As in the 2D version of the application, the pendulum's simulation is implemented inside the draw() function. Instead of using the canvas commands, the 3D shapes are being drawn based on the two array buffers holding the line and diamond shapes. The application uses the angles θ, ϕ and the positions (x1, y1, z1), (x2, y2, z2) as described in section 6.2.3.2 to place each pendulum in the correct position each time the draw() method is being executed.

To display the pendulum's state inside the panel, the following operations are being executed:

- Clear the canvas element
- Translate the line buffer in the centre of the canvas panel
- Rotate the line around the Z axis using the angle θ
- Draw the line
- Translate the diamond buffer in position (x1, y1, z1)
- Rotate the diamond around the Z axis using the angle θ
- Draw the diamond
- Translate the line buffer in position (x1,y1, z1)
- Rotate the line around the Z axis using the angle θ
- Rotate the line around the Y axis using the angle ϕ
- Draw the second line
- Translate the diamond buffer in position (x2, y2, z2)
- Rotate the diamond around the Z axis using the angle θ
- Rotate the line around the Y axis using the angle ϕ
- Draw the second diamond

Note that the matrix values after each transformation are stored in a different array so that the initial details of the shape remain the same and can be reused. For example in this application, the same line and diamond buffers are used for the generation of the top and the bottom pendulums.

The four arrow buttons located at the right bottom corner of the simulation's panel, allows the user to change the viewing angle of the scene. Each time one of the buttons is being clicked, the matrix holding the information of all the shapes in the scene is being rotated by 15° around the relevant axis. More precisely if the user clicks on the

top arrow button, the scene is rotated around the X axis by 15°. If users click on the bottom arrow button the scene is rotated by -15° around the X axis. Similarly, the left and right arrow buttons rotate the scene by 15° and -15° respectively around the Y axis. The auto-rotate button available in-between the arrow buttons, sets the pendulum to be rotated by 1° around The Y axis each time the draw() function is being executed.

The images below display the same state of the pendulum in different viewing angle selections. The first image displays the pendulum from an upper position, the second image displays the pendulum from the front position and the third image displays the pendulum from the right position.

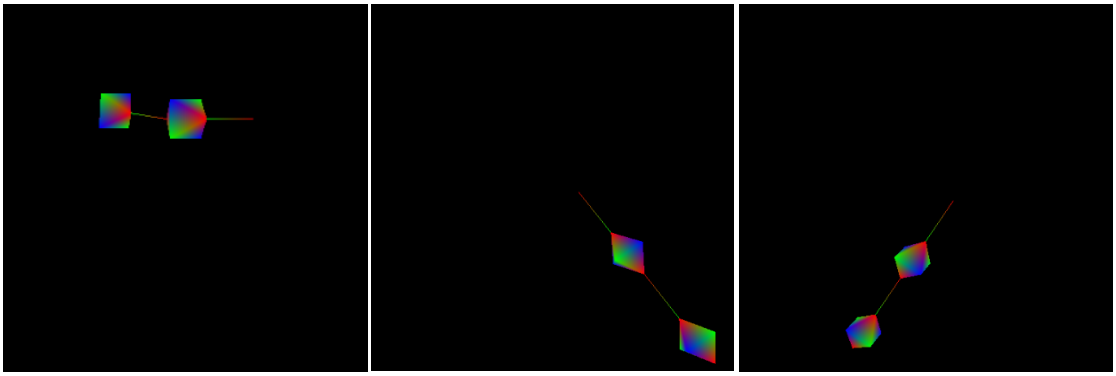


Figure 6.17: Double pendulum application (3D) – different viewing angles

### 6.2.5. Graph Plotter

Each application consists of one or two graph plotters used to represent the behaviour of the different simulation attributes. This chapter describes the functionality behind the interface of the graph plotter. An empty plotter is shown in the figure below.

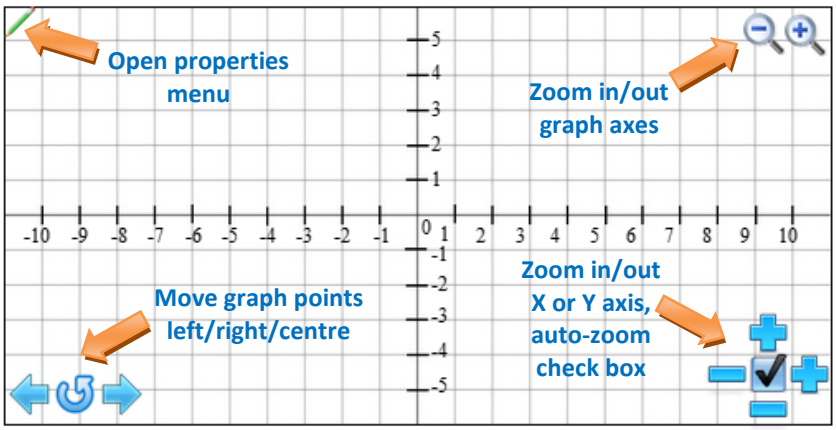



Figure 6.18: Graph plotter

The icon in the upper left corner of the plotter is used to open the properties menu. If the user moves the mouse pointer over the icon, the properties menu appears inside the plotter as shown below.

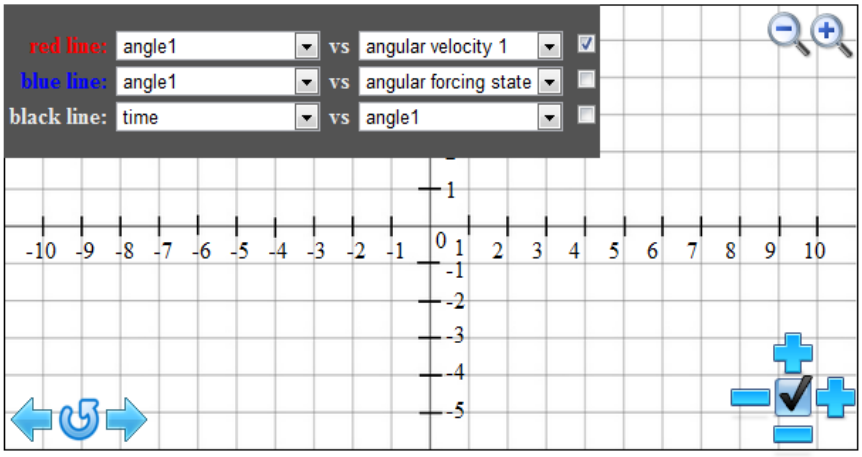



Figure 6.19: Graph plotter – properties menu

From the properties menu, users are able to choose what values will be represented, by each graph colour. In addition, using the check boxes in the right side of the menu, they can set which graphs will be displayed in each plotter.

During the run of the simulation, the selected values are being drawn inside the plotter as illustrated in the next figure.

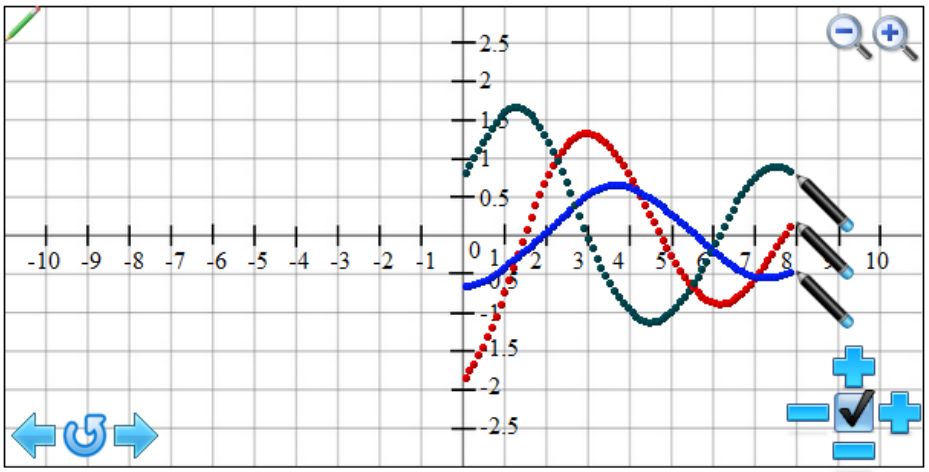


Figure 6.20: Graph plotter during the simulation

Each time the draw() function of the application is being executed, a new point is added on the plotter for each graph line. This is handled by another function called addPoint() which is invoked from the draw() function. The addPoint function takes three parameters. The first two parameters are the coordinates of the point on the X

and Y axes while the third parameter is the colour of the point to be added. The required values to be represented in each axis by each graph colour are taken from the user's selections in the graph properties menu.

Each new point on the graph is represented by a new <img> element. The source of the image element is set to a red, blue or black dot image depending on the graph line which is being used on. The position of the new element is set using two functions called xToPoint and yToPoint which translate the X and Y axis coordinates of the point into top and left pixel positions on the screen. Each point is generated and displayed inside the plotter dynamically. The first point of each graph is represented using a different "pencil" icon. This is due to the fact that in some graph representations the new points are placed on top of previous points, and thus, it may be hard to recognise the current state of the system.

The buttons placed on the right side of the panel are used to perform zoom operations on the current points of the graph. The two upper right buttons apply zoom in and zoom out operation on both axes as shown by the figures below.

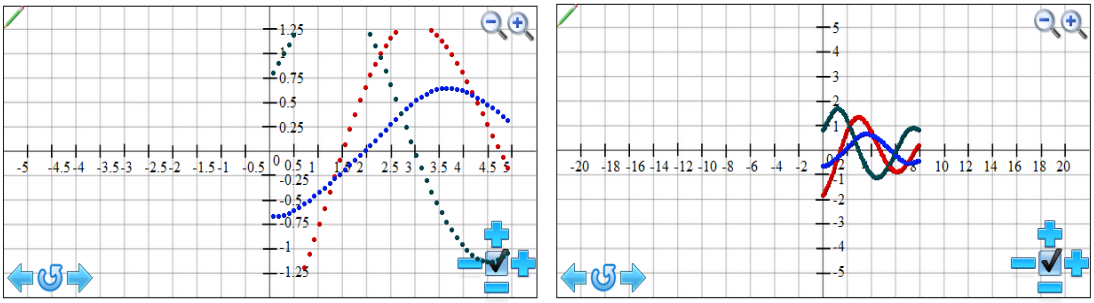


Figure 6.21: Graph plotter after zoom-in (left) and zoom-out (right) operations

The four "plus" and "minus" style buttons on the bottom right corner of the panel are used to perform zoom operations on only one of the two axis. The top "plus" and lower "minus" buttons perform zoom-in and zoom-out operations respectively on the Y axis. Similarly the right "plus" and left "minus" buttons perform zoom-in and zoom-out operations respectively on the X axis. The graph results after each operation are shown in the following page.

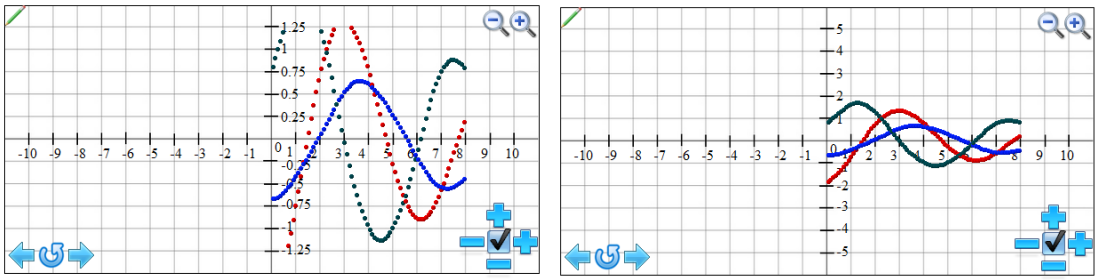


Figure 6.22: Graph plotter after zoom-in on Y axis (left) and zoom-out on Y axis (right) operations

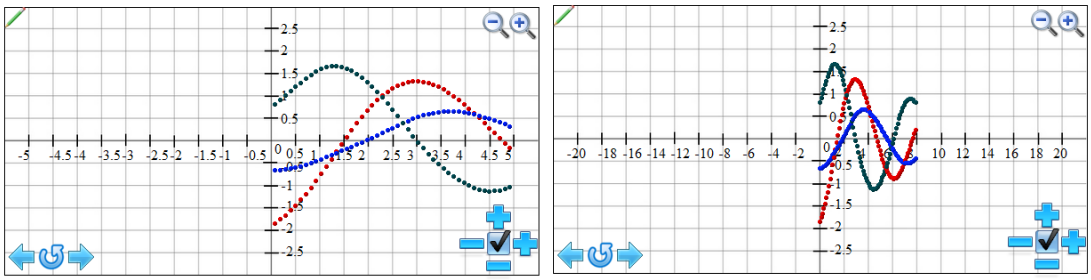


Figure 6.23: Graph plotter after zoom-in on X axis (left) and zoom-out on X axis (right) operations

All the zoom operations are handled by two functions called graph_zoom() and fix_nums() which are responsible to adjust the point positions and the graph coordinate numbers on each axis respectively. This is done using two variables "zoomFactorX" and "zoomFactorY" to hold the current zoom factors on both axes. The code of graph_zoom() and fix_nums() functions are available in appendix A.5.

The checkbox operation located in the centre of the four "plus" and "minus" buttons is used to adjust the "zoomFactor" value on each axis automatically so that the graph points are always fitted on the plotter's panel. The graph_zoom() and fix_nums() functions are being invoked every time the "zoomFactorX" or "zoomFactorY" value is has been modified. Below is an example of the same graph generated while the auto-zoom option was disabled (left figure) and enabled (right figure).

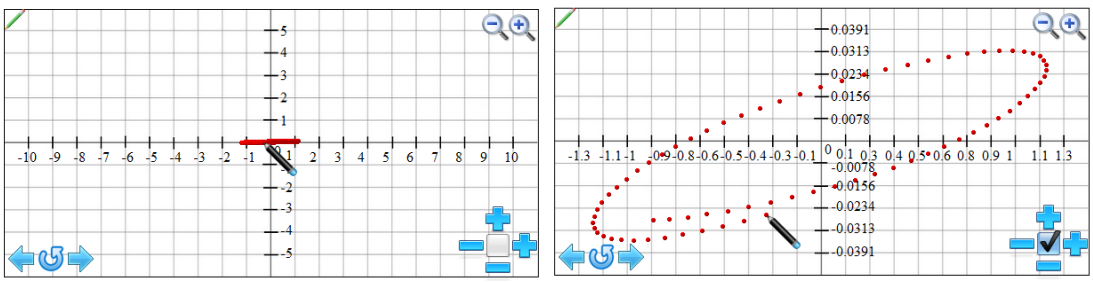


Figure 6.24: Graph plotter – auto-zoom disabled (left) and auto-zoom enabled (right)

If all the graphs are set to represent the time value on the X axis, when the time value becomes equal to maximum coordinate number of the axis, the graphs points and

coordinate values are being moved to the left such that the most recent point of the graph is placed on the middle of the panel's width as shown in the figure below.

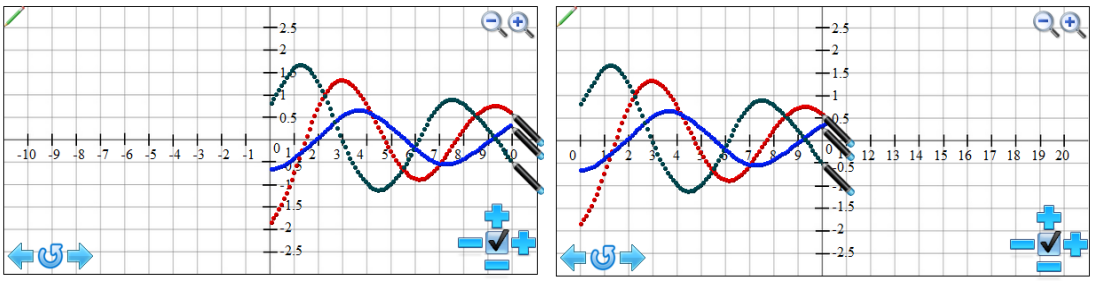


Figure 6.25: Graph plotter – move points on X axis

This is to let the new graph points to be visible inside the panel. The three buttons located on the bottom left corner of the panel are used to navigate the graph along the X axis and view previous points which are not currently displayed in the panel. The left button moves the graph points and the coordinate numbers of the X axis to the left; the right button moves them to the right and the centre button resets the coordinates of the graph to the origin of the axes.

# 7. System tests

## 7.1. Load application

| Test Case | Action performed | Required system's response | Actual system's response |
|---|---|---|---|
| Load web application | User enters the URL of the application's web page | Display application's main page | Main page displayed inside the browser |
| Load dimension reduction application | User clicks on the dimension reduction application link from the main menu | Display the initial page of the dimension reduction application | Initial Page of the application displayed |
| Load single pendulum application | User clicks on the single pendulum application link from the main menu | Display the initial page of the single pendulum application | Initial Page of the application displayed |
| Load double pendulum (2D) application | User clicks on the double pendulum (2D) application link from the main menu | Display the initial page of the double pendulum (2D) application | Initial Page of the application displayed |
| Load double pendulum (2D) application | User clicks on the double pendulum (3D) application link from the main menu | Display the initial page of the double pendulum (3D) application | Initial Page of the application displayed |

## 7.2. Dimension Reduction application

| Test Case | Action performed | Required system's response | Actual system's response |
|---|---|---|---|
| Add new point | User clicks on the graph coordinates | Display the new point on the graph, display the projection line inside the graph, display the calculation details on the right side of the page | Operations performed as required |
| Remove existing point | User clicks on an existing point | Display the graph without the clicked point, display the new projection line inside the graph, display the new calculation details on the right side of the page | Operations performed as required |

| | | | |
|---|---|---|---|
| View coordinates of an existing point | User moves mouse pointer over an existing point | Display the point's coordinates on the right side of the point | Coordinates displayed as required |
| Zoom in graph | User clicks on the zoom-in button | Display focused graph view | Focused graph view displayed |
| Zoom out graph | User clicks on the zoom-out button | Display wider graph view | Wider graph view displayed |

## 7.3. Single pendulum application

| Test Case | Action performed | Required system's response | Actual system's response |
|---|---|---|---|
| Start simulation | User clicks on the start button | Display the simulation inside the canvas panel, draw selected graphs inside the plotter | Operations performed as required |
| Stop simulation | User clicks on the stop button | Stop the simulation inside the canvas panel, stop drawing on the graph plotter | Operations performed as required |
| Change initial conditions | User edit the value in of an initial condition input box and clicks enter key | Display the new simulation inside the canvas panel, draw new graph values inside the plotter | Operations performed as required |

## 7.4. Single pendulum application

| Test Case | Action performed | Required system's response | Actual system's response |
|---|---|---|---|
| Start simulation | User clicks on the start button | Display the simulation inside the canvas panel, draw selected graphs inside the plotter | Operations performed as required |

| | | | |
|---|---|---|---|
| Stop simulation | User clicks on the stop button | Stop the simulation inside the canvas panel, stop drawing on the graph plotter | Operations performed as required |
| Change initial conditions | User edit the value in of an initial condition input box and presses the enter key | Display the new simulation inside the canvas panel, draw new graph values inside the plotter | Operations performed as required |

## 7.5. Double pendulum application (2D)

| Test Case | Action performed | Required system's response | Actual system's response |
|---|---|---|---|
| Start simulation | User clicks on the start button | Display the simulation inside the canvas panel, draw selected graphs inside the plotter | Operations performed as required |
| Stop simulation | User clicks on the stop button | Stop the simulation inside the canvas panel, stop drawing on the graph plotter | Operations performed as required |
| Restart simulation | User clicks on the restart button | Display the simulation from the initial state of the pendulum, draw selected graphs from the initial state of the pendulum | Operations performed as required |
| Change initial properties | User edit the value in of an initial property, in the initial properties' menu and presses the enter key | Display the new simulation inside the canvas panel, draw new graph values inside the plotter | Operations performed as required |
| Change initial condition | User edit the value in of an initial condition value from the initial condition's menu and presses the enter key | Display the new simulation inside the canvas panel, draw new graph values inside the plotter | Operations performed as required |

| | | | |
|---|---|---|---|
| Move pendulum simulation on its previous state | User clicks on the left arrow button inside the canvas panel | Display the previous state of the simulation inside the canvas panel, move the pointer of the graph plotter to point on the previous state of the simulation | Operations performed as required |
| Move pendulum simulation on its next state | User clicks on the right arrow button inside the canvas panel | Display the next state of the simulation inside the canvas panel, draw the new points inside the graph plotter representing the new state of the pendulum | Operations performed as required |

## 7.6. Double pendulum application (3D)

| Test Case | Action performed | Required system's response | Actual system's response |
|---|---|---|---|
| Enable autorotation of the 3D pendulum simulation | User clicks on the auto-rotate button inside the simulation panel | Display the simulation rotated by 1° around the Y axis every time a new state of the pendulum is being shown | Operation performed as required |
| Disable autorotation of the 3D pendulum simulation | User clicks on the disable auto-rotate button inside the simulation panel | Display the simulation with no rotations | Operation performed as required |
| Change the viewing angle of the simulation towards the top side of the pendulum | User clicks on the top rotate button inside the panel | Display the simulation rotated by 15° around the X axis | Operation performed as required |
| Change the viewing angle of the simulation towards the bottom side of the pendulum | User clicks on the bottom rotate button inside the panel | Display the simulation rotated by -15° around the X axis | Operation performed as required |

| | | | |
|---|---|---|---|
| Change the viewing angle of the simulation towards the right side of the pendulum | User clicks on the right rotate button inside the panel | Display the simulation rotated by 15° around the Y axis | Operation performed as required |
| Change the viewing angle of the simulation towards the left side of the pendulum | User clicks on the left rotate button inside the panel | Display the simulation rotated by -15° around the Y axis | Operation performed as required |

## 7.7. Graph plotter

| Test Case | Action performed | Required system's response | Actual system's response |
|---|---|---|---|
| Change the values described on each axis of the graph for each line colour | User chooses new value(s) to be described on the graph axes using the graph's properties menu | Draw the values selected to be displayed by each graph line on the plotter | Operation performed as required |
| Enable graph appearance | User checks the check-box near the graph lines required to be displayed inside the plotter | Draw the required lines inside the graph | Operation performed as required |
| Disable graph appearance | User uncheck the check-box near the graph lines not required to be displayed inside the plotter | Draw only the required lines inside the graph | Operation performed as required |
| Enable auto-zoom property | User checks the auto-zoom check-box | Display the graph lines adjusted to fit the graph's size | Operation performed as required |
| Disable auto-zoom property | User uncheck the auto-zoom check-box | Display the graph lines using the initial coordinates of the graph plotter | Operation performed as required |

| | | | |
|---|---|---|---|
| Zoom-in graph coordinates | User clicks on the zoom-in button | Display a focused view of the graph points and coordinate numbers | Operation performed as required |
| Zoom-out graph coordinates | User clicks on the zoom-out button | Display a wider of the graph points and coordinate numbers | Operation performed as required |
| Zoom-in graph coordinates on X axis | User clicks on the top plus button | Stretch the view of the graph points and coordinate numbers on the X axis | Operation performed as required |
| Zoom-out graph coordinates on X axis | User clicks on the bottom minus button | Reduce the view size of the graph points and coordinate numbers on the X axis | Operation performed as required |
| Zoom-in graph coordinates on Y axis | User clicks on the top right button | Stretch the view of the graph points and coordinate numbers on the Y axis | Operation performed as required |
| Zoom-out graph coordinates on Y axis | User clicks on the left minus button | Reduce the view size of the graph points and coordinate numbers on the Y axis | Operation performed as required |
| Move graph coordinates towards the left side of the plotter | User clicks on the left arrow button | Display graph points and X coordinate numbers shifted towards the left side of the graph | Operation performed as required |
| Move graph coordinates towards the right side of the plotter | User clicks on the right arrow button | Display graph points and Y coordinate numbers shifted towards the right side of the graph | Operation performed as required |
| Reset graph's view to the origin of the axes | User clicks on the reset button | Reset graph move operations and display the origin of the axis on the centre of the graph plotter | Operations performed as required |

| | | | |
|---|---|---|---|
| Clear graph points | Uses clicks on the clear button inside the simulation panel | Display the graph plotter without any of the previously added points | Operation performed as required |

# 8. System evaluation

Two evaluation sessions took place after the implementation of the system in order to ensure the quality of the application. In the first session, the application has been evaluated by people who were aware of the project and have seen the application at different stages during its implementation. One the people, who were present in this session, was the creator of the two simulation programs used in this project. In the second evaluation session, the application has been evaluated by a person who had never seen the application before the session. In both evaluations, users have been provided with the user manual of the implemented application. However, users were able to apply most of the operations available in the user interface of the application before even reading the manual.

The quality of the application in each aspect is described in the following table.

| Aspect | Quality |
|---|---|
| Functionality | All the functions of the application have been tested using the test sets described in the previous chapter. All operations returned the correct response. |
| Maintainability | The system has been implemented using 3-tier architecture. Thus it provides the ability for maintenance to be applied in any of the three tiers (presentation, business, and data) without affecting the other ones. The system has been designed in such way that any future dynamical systems can be added efficiently. The interface of the application provides all the features that ay dynamical system may require to describe its behaviour, i.e. 2D representations, 3D representations and an interactive graph plotter. Thus any new dynamical system can be added on the web application by using a new executable file on the server and adjusting the user interface of the application to display the activity of the new dynamical system. |
| Communication model | The communication between client and server sides is performed using the XMLHttpRequest of Ajax. |
| Portability | The system can be installed on both PHP and JSP servers since it provides the same functionality in two different script languages. |

| | |
|---|---|
| Usability | Most buttons have been represented using different icons in order to help users understand the action of each button. |
| Efficiency | Executable programs may need more than 10 seconds to provide the required output values. However using asynchronous XMLHttpRequest, the application manages to start the simulation's representation before the end of the execution on the server-side, using only the first output values of the execution. The application needs approximately 2 seconds to start the simulation after user request. |
| Security | The application is not intended to contain any valuable information, thus, security was not one of the requirements of the system. However it is better to be included in an existing web interface where only members of CICADA have access. |
| Reliability | The dynamical system's representations and the values displayed inside the application have been checked against the actual output of the executable programs to ensure that the correct information is represented by the application. The interface information was correct. |

# 9. Limitations – Future work

During the evaluation sessions, users introduce the requirement for graph plotters to be able to draw direct lines instead of multiple points. The is due to the behaviour of different graphs which keep drawing the line using a similar path and thus, after many point additions it is difficult to recognise which points are for each line. The reason that the graph plotter has been initially designed to draw points instead of lines was to represent on the output values as they have been taken from the output of the server program's execution. It would be nice, as additional functionality, the system to include an option letting the user to choose if he/she wants the plotter to display lines or points inside the graph. However even using lines, sometimes it may be still hard to recognise all lines if they are very close to each other. Another, maybe easier, way to overcome this situation is to set the "clear graph" command to be executed after some fixed time intervals. In this way, the graph will always display the more recent values of the simulation.

Another limitation is the fact that WebGL is not supported by older versions of internet browsers. This is unavoidable since WebGL is a recent technology. However it would not be very difficult for users to update the version of internet browsers they are using. The fact that the first specification of WebGL has been released on February of 2011 [40] and is already being supported by every commonly used browser, except from Internet Explorer, and even by some mobile devices, is very promising for its future support in all browsers and operating systems.

Another interesting functionality to be added on the current version of the application is the relation of the dimension reduction application and the dynamical system applications. More precisely it would be nice if values from the simulations could be used on the dimension reduction application to display the projection of these values in one dimensional space. The dimension reduction application can be also modified to allow more advanced reductions to be made, taking as inputs points of higher dimensional spaces.

Moreover, as future work on the system must be considered the representation of new dynamical systems using the provided interface and functionality. As mentioned earlier the application has been designed to adopt any additional dynamical system.

# 10. Project plan

In general all the phases of the project have been completed successfully. However the actual timing was not as set in the project plan at the beginning of the project (see appendix B). The implementation phase took longer than expected and, thus, the testing and evaluation phases had to be completed in less time than they were planned. However this time was enough to complete the implementation of the system, apply the relevant unit tests, attend in two evaluation sessions and write the final report for the project.

# 11. Summary

Inside this report it has been described how the project has been designed, implemented and tested in order to meet the project's requirements. The application has been designed as a framework to represent the behaviour of any dynamical system inside a web browser. The web application uses rendering technologies to display the simulations of the dynamical systems in both 2D and 3D representations. Furthermore, a graph plotter is used to describe the behaviour of the different dynamical systems during the simulation.

The web application is available at http://nikniknik75.s20.eatj.com

The web page includes a link to the user manual of the application as well as information on how to enable WebGL in each browser.

# References


[1] Scheinerman, E.R.(1996) *Invitation to Dynamical Systems*, Michigan: Prentice Hall

[2] Lygeros, J. (2004) 'Lecture Notes on Hybrid Systems', Available at: http://robotics.eecs.berkeley.edu/~sastry/ee291e/lygeros.pdf, Last Accessed: 7th May 2011

[3] Klages, R. (2008) 'Introduction to Dynamical Systems: Lecture Notes for MAS424/MTHM021', Available at: http://www.maths.qmul.ac.uk/~klages/mas424/lnotes_ds2007f.pdf, Last Accessed: 6th May 2011

[4] Tomlin, C.J. (2005) 'AA278A Lecture 1: Introduction', Available at: http://www.stanford.edu/class/aa278a/lecture1.pdf, Last Accessed: 7th May 2011

[5]http://www.cicada.manchester.ac.uk/research/human_falling/index.php, CICADA: Human Falling

[6] http://www.maths.manchester.ac.uk/~shmerkin/#exampleproblem, Pablo Shmerkin: An example problem

[7] http://www.britannica.com/EBchecked/topic/1366374/client-server-architecture, client-server architecture in Encyclopaedia Britannica online

[8] Berners-lee, T. and Fischetti (1999) *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*, San Francisco: HarperCollins

[9] Cormode, G. and Krishnamurthy, B. (2008) 'Key Differences between Web1.0 and Web2.0', *First Monday* 13(6):1-30

[10] http://oreilly.com/web2/archive/what-is-web-20.html, What Is Web 2.0: Design Patterns and Business Models for the Next Generation of Software in O' REILLY, Sept 2005

[11] Crane, D., Pascarello, E. and James, D. (2006) *Ajax in action*, Greenwich: Manning Publications

[12] http://www.caritech.com/node/96, Getting started with Ajax, Jan 2009

[13] https://developer.mozilla.org/en/xmlhttprequest, XMLHttpRequest in MDC, Apr 2011

[14] http://www.w3.org/TR/XMLHttpRequest, XMLHttpRequest W3C specification, Aug 2010

[15] http://xul-fr.org/wiki/ApplisWeb/Request, XMLHttpRequest, 2009

[16] http://www.w3schools.com/css, CSS Tutorial, Apr 2011

[17] http://www.w3.org/DOM/, Document Object Model (DOM) W3C specification, Jan 2005

[18]Cerami, E. (2002) *Web Services Essentials*, CA: O'Reilly Media

[19] http://www.w3schools.com/soap/soap_intro.asp, SOAP Introduction, Jun 2003

[20] http://www.w3schools.com/wsdl/wsdl_intro.asp, Introduction to WSDL, Jun 2007

[21]http://www.ajaxonomy.com/2008/xml/web-services-part-1-soap-vs-rest, Web Services, Part 1: SOAP vs. REST, May 2008

[22] Newcomer, E. (2002) *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Boston: Addison-Wesley Professional

[23] http://www.w3schools.com/wsdl/wsdl_ports.asp, WSDL Ports, Apr 2011

[24] http://www.w3schools.com/wsdl/wsdl_binding.asp, WSDL Bindings, Apr 2011

[25] http://diveintohtml5.org/canvas.html, Canvas element description

[26] http://code.google.com/p/explorercanvas, Explorer Canvas

[27] https://developer.mozilla.org/en/Canvas_tutorial/Basic_usage, Canvas basic usage, May 2011

[28]https://developer.mozilla.org/en/Canvas_tutorial/Drawing_shapes, Canvas shapes, Jul 2011

[29] http://code.google.com/apis/o3d, O3D plugin API, 2011

[30] https://wiki.mozilla.org/Canvas:3D, Canvas:3D, Jan 2010

[31]http://www.khronos.org/webgl, WebGL - OpenGL ES 2.0 for the Web, 2011

[32]http://www.khronos.org/opengl, OpenGL - The Industry's Foundation for High Performance Graphics, 2011

[33]https://developer.mozilla.org/en/WebGL/Adding_2D_content_to_a_WebGL_context, Adding 2D content to a WebGL context – WebGL initialization functions, Mar 2011

[34]https://developer.mozilla.org/en/WebGL/Getting_started_with_WebGL, Getting started with WebGL from Mozilla, Jul 2011

[35]http://www.khronos.org/webgl/wiki/Tutorial, Khronos WebGL tutorial, March 2011

[36] http://learningwebgl.com/blog/?p=1253, spheres, rotation and mouse events in WebGL

[37]http://www.opengl.org/sdk/docs/man/xhtml/glDrawArrays.xml, glDrawArrays in OpenGL, 2006

[38] Skeldon, A. C. (1994) Dynamics of a parametrically excited double pendulum

[39] http://code.google.com/p/glmatrix, glmatrix project home

[40] https://www.khronos.org/registry/webgl/specs/1.0, Khronos WebGL 1.0 specification

## Appendix A

### 1. Shaders

Vertex shader example

```
<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;

  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;

  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
  }
</script>
```

Fragment shader example

```
<script id="shader-fs" type="x-shader/x-fragment">
  void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }
</script>
```

### 2. pointToX()and pointToY() functions

```
function pointToX(x){
 var pos_x=((27.3*((x)*zoomfactorx+10))+26);
 return pos_x;
}

function pointToY(y){
  var pos_y=((27.4*(y*zoomfactory*(-1)+10))+26);
  return pos_y;
}
```

### 3. addPoint() function

```
function addPoint(event){
    var pos_x = event.offsetX?(event.offsetX):event.pageX-document.getElementById("pointer_div").offsetLeft;
    var pos_y = event.offsetY?(event.offsetY):event.pageY-document.getElementById("pointer_div").offsetTop;

    var newdivpoint = document.createElement('div');
    var pointdivid = 'pointdiv'+pointnum;
    var newpoint = document.createElement('img');
    var pointid = "point"+pointnum;
    newpoint.setAttribute('id',pointid);
    newpoint.setAttribute('src',"point.gif");
    newpoint.setAttribute('style',"position:absolute");

    if(navigator.appName!="Microsoft Internet Explorer"){
    var mouseevent = "removePoint('"+pointid+"',event);";
    newpoint.setAttribute('onmouseup',mouseevent);
    }   else
    newpoint.onmouseup = function() {removePoint(pointid,event);};
    newpoint.onmouseover = function() { viewPoint(pointid,pos_x,pos_y);};
    newpoint.onmouseout = function() { hidePoint(pointid); };
    newpoint.setAttribute('class','dragme');

    var graph = document.getElementById("pointer_div");
    graph.appendChild(newpoint);
    document.getElementById(pointid).style.zIndex=2;
    document.getElementById(pointid).style.left = (pos_x-5) ;
    document.getElementById(pointid).style.top = (pos_y-5) ;

    pointnum++;

    pointarray_x.push(xtopoint(pos_x));
    pointarray_y.push(ytopoint(pos_y));

    add();

}
```

### 4. draw() function

```
    function draw(){

    if (response.length<100)
    response=response+tempresponse;


    if(t<=20){
    addPoint(t,a,"red");
    addPoint(t,v,"blue");
    addPoint(t,ac,"black");
    }

        x = 200*Math.sin(a);
    y = 200*Math.cos(a); //sin(180-90-a0);


    //make pendulum line
    ctx.lineWidth = 10;
    ctx.lineCap ='round';
    ctx.beginPath();
    ctx.moveTo(0,0);
    ctx.lineTo(x,y);
    ctx.stroke();

    //make pendulum ball
    ctx.beginPath();
        ctx.arc(x,y,30,0,Math.PI*2,0); // Outer circle
    ctx.fill();
    ctx.restore();

    //ctx.translate(-300,-300);


      }
```

## 5. graphzoom() function

```
function graphzoom(act){

    if (act=="plus"){
    zoomfactorx*=2;
    zoomfactory*=2;
    }
    else if(act=="minus"){
...

    var max=pointnum;
    pointnum=0;

    for (var i=0; i<max; i++){
    var pointid="point"+i;
    var point=document.getElementById(pointid);
    var graph=point.getAttribute("graph");

    if(graph=="graph1"){
    var x=point.getAttribute("xpos");
    var y=point.getAttribute("ypos");
    var color=point.getAttribute("color");

    var graph=document.getElementById("graph1");
    graph.removeChild(point);
    addPoint(x,y,color,true,"graph1");
    }
    else{
    pointnum++;
    }   }

    fixnums();

}
```

## 6. fixnums() function

```
function fixnums(){

    var valuey=5;
    var valuex=-10;
    for(var i=1; i<21; i++){

    var divxid="divx"+i;
    var divyid="divy"+i;

    if(valuex==0)
    valuex++;

    var x=(valuex-movementx)/zoomfactorx;
    if(x.toFixed(1)!=x) x=x.toFixed(1);
    document.getElementById(divxid).innerHTML=x;
    if(valuey==0)
    valuey--;

    var y=valuey/zoomfactory;
    if(y.toFixed(4)!=y) y=y.toFixed(4);
    if(i<=10)
    document.getElementById(divyid).innerHTML=y;

    valuex++;
    valuey--;

    }

}
```

## Appendix B

| | Task Name | Duration | Start | Finish |
|---|---|---|---|---|
| 1 | **Project** | **213 days** | **Tue 01/02/11** | **Thu 01/09/11** |
| 2 | **Research** | **59 days** | **Tue 01/02/11** | **Thu 31/03/11** |
| 3 | Literature Survey | 28 days | Tue 01/02/11 | Mon 28/02/11 |
| 4 | Background research | 31 days | Tue 01/03/11 | Thu 31/03/11 |
| 5 | Finish research | | | Thu 31/03/11 |
| 6 | **Simple implementation** | **38 days** | **Fri 01/04/11** | **Sun 08/05/11** |
| 7 | Imlement simple hybrid systems using AJAX | 25 days | Fri 01/04/11 | Mon 25/04/11 |
| 8 | Report | 13 days | Tue 26/04/11 | Sun 08/05/11 |
| 9 | Finish implementation | | | Sun 08/05/11 |
| 10 | **Product specification** | **23 days** | **Mon 09/05/11** | **Tue 31/05/11** |
| 11 | Requirements analysis for CICADA developers | 23 days | Mon 09/05/11 | Tue 31/05/11 |
| 12 | Finish analysis | | | Tue 31/05/11 |
| 13 | **Implementation** | **30 days** | **Wed 01/06/11** | **Thu 30/06/11** |
| 14 | Prototype implementaion | 30 days | Wed 01/06/11 | Thu 30/06/11 |
| 15 | Finish prototype | | | Thu 30/06/11 |
| 16 | **Testing** | **20 days** | **Fri 01/07/11** | **Wed 20/07/11** |
| 17 | Evaluation | 10 days | Fri 01/07/11 | Sun 10/07/11 |
| 18 | Unit testing | 10 days | Mon 11/07/11 | Wed 20/07/11 |
| 19 | Finish testing | | | Wed 20/07/11 |
| 20 | Final Product | 11 days | Thu 21/07/11 | Sun 31/07/11 |
| 21 | Project Report | 31 days | Mon 01/08/11 | Wed 31/08/11 |
| 22 | End of project | | | Thu 01/09/11 |
| 23 | | | | |
| 24 | | | | |